FRANTCHESCO CECCHIN

UM MODELO PARA RESOLUÇÃO DE CONFLITOS SOBRE REPOSITÓRIO DE DADOS XML

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora: Prof[a]. Dr[a]. Carmem Satie Hara

CURITIBA

2010

Cecchin, Frantchesco
Um modelo para resolução de conflitos sobre repositório de dados XML / Frantchesco Cecchin. - Curitiba, 2010.
92 f. : il., tabs.

Dissertação (Mestrado) – Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.
Orientadora: Carmem Satie Hara
1. XML (Linguagem de marcação de documento). 2. Limpeza de dados. 3. Banco de dados. I. Hara, Carmem Satie. II. Título. III. Universidade Federal do Paraná.

CDD 005.72

Agradecimentos

Primeiramente, eu gostaria de agradecer a minha orientadora Carmem Satie Hara, sem a qual essa dissertação nunca teria começado e o que é pior, nunca teria terminado. Agradeço imensamente por acreditar neste trabalho e guiar-me pelo árduo caminho do mestrado.

A professora Dr^{a}. Cristina Dutra de Aguiar Ciferri por sua inestimável contribuição no momento da escrita do artigo e pelas discussões e revisões efetuadas, inclusive nos finais de semana.

Ao colega Bruno Tomazela, que apesar de são-paulino, sempre atendeu prontamente minhas solicitações e questionamentos. Estendo este agradecimento a todos os colegas do mestrado na UFPR que compartilhavam as mesmas dificuldades.

A minha namorada Suélyn pela sua paciência e dedicação. Embora, tenha repetido invariavelmente a pergunta proibida "quando você vai acabar esse mestrado?" diante das minhas negativas em ir ao cinema ◡̈.

Por fim, agradeço a minha amada família, as pessoas que mais torceram, mais acreditaram e que sempre me apoiaram. Minha irmã Analice pelo carinho. Meus pais Ulisses e Eli Ana Cecchin que mesmo geograficamente distante, sempre estiveram muito presentes, pais que me ensinaram o valores da vida e não mediram esforços para que eu chegasse até aqui.

Obrigado a todos! Vocês são parte desta dissertação de mestrado.

"Um sonho que se sonha só, é só um sonho que se sonha só, mas sonho que se sonha junto é realidade."

Raul Seixas

Resumo

Garantir a qualidade dos dados quando se deseja manter informações provenientes de fontes heterogêneas é um desafio. Os dados importados destas fontes podem conter redundâncias, inconsistências ou ainda estar estruturados de formas completamente distintas. Existem diversas formas de melhorar a qualidade dos dados, tais como realizar bons mapeamentos entre fontes e repositório, identificar objetos semelhantes e manter uma única representação do dado.

Para este trabalho, considera-se que questões como mapeamentos, integração e detecção de duplicidade já foram resolvidos. Desta forma, o modelo proposto tem seu foco no estágio subsequente, ou seja, a resolução dos conflitos gerados pela integração. A abordagem para resolução de conflitos considerada tem como base a aplicação de uma política de fusão. Esta política é uma composição de regras definidas pelo usuário para solucionar os conflitos em determinado contexto do repositório. Tais regras têm o objetivo de representar as decisões que o usuário toma quando realiza a limpeza manualmente. Desta forma, uma vez que a regra foi definida, os conflitos reincidentes são solucionados automaticamente nas integrações futuras. Além disso, o modelo proposto considera um histórico de resoluções para manter a proveniência dos dados descartados e permitir auditar as decisões aplicadas. A manutenção da proveniência permite ao modelo reconstruir a fonte de dados original, evitando o armazenamento de uma cópia das mesmas.

Para validar o modelo foi desenvolvida uma ferramenta, denominada XFusion, a qual permitiu executar todas as funcionalidades do modelo sobre um repositório integrado de dados. Adicionalmente, testes de desempenho foram executados e os resultados obtidos mostram a viabilidade do modelo.

Abstract

Ensuring high quality data when collecting and integrating information from heterogeneous sources into a data warehouse is a challenging problem. In this master thesis, we propose a model for XML data integration, which allows the integrator to define data cleaning rules for solving value conflicts that may have been detected during the integration process. These rules resemble decisions that are made by users when data are manually curated and, once defined, conflicts detected in subsequent integration processes that are within the context of existing rules can be automatically solved without user intervention. Moreover, the proposed model maintains a resolution log for storing provenance information of discarded data and enable us to audit prior decisions. The maintenance of provenance allows the model to reconstruct the original data source, avoiding the need to maintain local copies.

To validate our proposal, we developed XFusion, a tool that stores data integrated according to cleaning rules in a curated repository. Additionally, our experimental study shows the viability of implementing the model.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo

1

Introdução

Empresas de todos os tamanhos e de diversos segmentos implementam e utilizam os benefícios proporcionados pela manutenção de um repositório integrado de dados. É cada vez mais claro que o armazenamento de dados de forma integrada é uma excelente plataforma para transformar a imensa quantidade de dados existentes nestas organizações em informação útil e confiável para responder suas questões e auxiliá-los nos processos de tomada de decisão. Um repositório de dados oferece uma base única para a aplicação das técnicas de análise de dados existentes atualmente, tais como mineração de dados e análise multidimensional, além das consultas tradicionais e da geração de relatórios.

Prover o acesso a múltiplas fontes de dados através de um acesso integrado é um desafio que tem despertado o interesse de pesquisadores na área de sistemas de informação. Nesse contexto, dois problemas são fundamentais. Primeiro, como determinar se as fontes de dados contêm informações semanticamente relacionadas, isto é, se elas fazem referência ao mesmo conceito no mundo real. Segundo, como tratar a heterogeneidade semântica a fim de dar suporte ao processo de integração e permitir interfaces de consultas uniformes. A resolução desses problemas contribui para a manutenção de dados consistentes e precisos em um repositório integrado.

Sendo a Internet um enorme celeiro de informações e com o XML tornando-se o formato preferido para transmissão desses dados na grande rede, muitos trabalhos buscam

melhorar a qualidade dos dados no formato XML. Além disso, é natural executar o processo de limpeza dos dados no próprio formato XML, uma vez que sua estrutura hierárquica representa naturalmente o relacionamento entre os dados. Ou seja, se os dados são mapeados para o modelo relacional, faz-se necessária a execução de junções para que seja possível obter a mesma visão de relacionamento proporcionada pela estrutura em forma de árvore XML.

Para mostrar a importância da utilização de dados integrados de forma eficiente, imagine que se deseja manter um repositório integrado com informações acadêmicas sobre currículos de pesquisadores, com a finalidade de medir seus índices de produtividade e também permitir que os pesquisadores possam atualizar seus currículos a partir do repositório. As informações podem ser obtidas de fontes como a Plataforma Lattes (CNPq, 2010) e a *Digital Bibliography & Library Project* (DBLP) (Ley, 2010). Para integrar esses dados, inicialmente define-se um mapeamento entre entidades descritas na fonte e suas correlatas no repositório. Com esse mapeamento é possível ter um resultado integrado. No entanto, o resultado dessa integração pode ser impreciso, apresentando dados redundantes ou com conflitos ainda não solucionados.

Para ilustrar a necessidade de um processo de limpeza em um cenário no qual os dados são mantidos em um repositório integrado, suponha que durante o processo de integração de uma nova fonte de dados descobre-se que o artigo X do pesquisador Y apresenta o local de publicação distinto entre o informado na fonte e armazenado no repositório. Logo, é necessário tomar uma decisão sobre qual valor a ser escolhido, caso contrário o repositório manterá ambos os dados, causando imprecisão no resultado das consultas executadas. Desta forma, é interessante que o estágio de limpeza dos dados ofereça um recurso que auxilie o usuário no momento de tomar esse tipo de decisão. Por exemplo, a utilização de estratégias, que baseadas em algum critério de decisão, possam ser aplicadas automaticamente e solucionar o conflito.

A fusão de dados é o processo de combinar múltiplas entidades que representam o mesmo objeto do mundo real em uma representação única, limpa e consistente. Para alcançar bons índices de resoluções automáticas e minimizar as intervenções humanas, esta

dissertação propõe um modelo de limpeza de dados baseado em uma política de fusão. Esta política é composta por regras definidas pelo usuário e que permitem solucionar conflitos em determinado contexto do repositório. Assim, sempre que houver reincidência de um conflito na mesma entidade do repositório, o sistema saberá como atuar e qual valor escolher. Por exemplo, imagine que em uma integração o usuário optou manualmente pelo valor `'Z'` como local de publicação do artigo X. Esta decisão é armazenada como uma regra da política e caso esse conflito volte a ocorrer, o sistema teria condições de solucioná-lo automaticamente baseado na decisão do passado.

A manutenção de informações sobre proveniência dos dados também apresenta benefícios para o modelo de integração apresentado nesta dissertação. O principal propósito das anotações de proveniência é permitir a identificação da origem de cada item de dado armazenado no repositório. Nesta dissertação essas anotações têm fundamental importância especialmente por dois motivos. Primeiro, sua utilização como forma de mensurar a qualidade e a quantidade de confiança que se pode atribuir aos dados de determinadas fontes. Este é um recurso muito útil para compor estratégias de resolução de conflitos. Segundo, a manutenção de informações sobre a origem do dado permite a reconstrução da porção da fonte que originou aquele dado. Assim, quando uma nova versão desta fonte estiver disponível para integração, é possível identificar e processar somente as alterações entre uma versão e outra, reduzindo os custos de processamento e melhorando o desempenho do processo de integração. Além disso, com a possibilidade de reconstrução da porção original da fonte, não é necessário manter cópias das fontes originais como forma de garantir os dados integrados.

1.1 Motivação

Uma das grandes vantagens proporcionadas pela manutenção de sistemas de dados integrados é que os usuários destes sistemas podem obter uma visão geral, completa e precisa dos dados sem a necessidade de consultar todas as fontes separadamente. É completa porque nenhum objeto é deixado de fora do resultado e é precisa porque os dados são apresentados sem repetições e sem contradições. Manter o repositório de dados com altos

índices de precisão é o grande desafio deste trabalho. Para isso, é proposto um modelo de resolução de conflitos em dados XML que permite ao usuário do repositório estabelecer uma política de governança sobre os dados. Isto é, baseado nas decisões comumente tomadas pelo usuário, a política estabelece regras de como os conflitos devem ser resolvidos em determinados contextos do repositório. Uma política bem definida reduz o esforço manual demandado pelo usuário no estágio de limpeza dos dados.

1.2 Contribuições

Este trabalho expressa sua contribuição na área de gerenciamento de dados XML, especificamente na integração e resolução de conflitos destes dados quando deseja-se mantê-los integrados em um repositório central. Neste contexto, as principais contribuições desta dissertação são:

- Um modelo para integração e limpeza de dados baseado na aplicação de regras para resolução de conflitos em nível de instância. Estas regras remetem às decisões comumente tomadas pelo usuário nas resoluções manuais e reduzem a intervenção humana no processo de limpeza dos dados.

- Um processo de reconstrução do documento integrado, com a estrutura e os dados originais. Além disso, permite que seja reconstruído um documento de sugestão, com a estrutura original, mas apenas com os dados considerados corretos no contexto do repositório integrado.

- Uma ferramenta gráfica, chamada XFusion, que permite administrar o repositório integrado de dados, integrar novas fontes, reconstruir fontes integradas e definir novas regras de resolução de conflitos quando estes ocorrem em contextos descobertos pela política de fusão.

1.3 Organização do Trabalho

O restante desta dissertação está dividido da seguinte forma: o Capítulo 2 apresenta o estado da arte em integração de dados, relacionais e XML, e problemas típicos em sistemas de integração de dados. No Capítulo 3 são discutidos os trabalhos relacionados e sistemas de resolução de conflitos em dados XML. A descrição do modelo para integração e limpeza de dados proposto nessa dissertação é apresentada no Capítulo 4. No Capítulo 5 é apresentada a ferramenta desenvolvida para validação do modelo e seus detalhes de implementação. A análise de resultados e os experimentos realizados são apresentados no Capítulo 6. Por fim, no Capítulo 7, são feitas as considerações finais e sugestões de trabalhos futuros.

Capítulo

2

Integração de Dados

A integração de dados proporciona a capacidade de manipular dados oriundos de múltiplas fontes de forma transparente (Cruz e Xiao, 2005). Esta possibilidade é especialmente útil em aplicações como comércio eletrônico, gerenciamento de informações médicas, geográficas ou mesmo de informações empresariais. Uma abordagem que auxilia nesta tarefa é a construção de um repositório de dados.

Alguns benefícios proporcionados pela manutenção de todos os dados em um único repositório são os seguintes:

- apresenta um modelo de dados comum para todos os dados de interesse, sem considerar a distribuição e estruturação original dos dados;
- a possibilidade de formular consultas combinando informações de diferentes fontes, mas com uma interface única;
- auxilia o processo de análise e divulgação de dados, uma vez que as inconsistências são identificadas e resolvidas antes do seu armazenamento no repositório;
- permite que os dados sejam armazenados de forma segura por um tempo determinado, mesmo que os dados sejam removidos de suas fontes, dado que um repositório de dados é controlado apenas pelos seus administradores.

2.1 Modelo de Integração

Nesta seção é apresentada uma visão geral sobre sistemas de integração de dados com relação à forma de descrever sua arquitetura.

Definição 2.1. Um sistema de integração de dados $\mathcal{D}$ consiste de um tripla $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, onde $\mathcal{G}$ refere-se ao esquema global do repositório, sua estrutura e restrições; $\mathcal{S}$ refere-se ao esquema das fontes de dados, sua estrutura e restrições; e $\mathcal{M}$ que representa o mapeamento entre $\mathcal{G}$ e $\mathcal{S}$. □

Existem duas propostas gerais para o problema de modelagem: a integração lógica e a integração física.

Na integração lógica pura, o esquema global é uma entidade estritamente virtual. As consultas realizadas sobre ele são dinamicamente reescritas em tempo de execução e redirecionadas para as fontes de origem dos dados. O resultado é coletado das fontes através de mediadores e combinado para ser apresentado ao usuário. Um mediador é uma camada intermediária entre os aplicativos do usuário e as fontes de dados. Quando o usuário deseja obter um dado, ele acessa a fonte original utilizando as funcionalidades oferecidas pelo mediador.

A integração física é fortemente relacionada com visões materializadas de banco de dados e consiste em primeiro armazenar e integrar os dados de todas as fontes para então realizar consultas sobre eles. Repositório de dados integrado é um exemplo bem conhecido desta forma e integração.

O foco deste trabalho é voltado para a integração física, que a partir deste momento será tratada simplesmente por integração de dados. A Figura 2.1 ilustra uma arquitetura clássica de um sistema de integração de dados, a qual emprega um esquema intermediário global a fim possibilitar a interoperação, via mapeamentos, com as diversas fontes autônomas. A partir de uma aplicação as consultas são efetuadas sobre o repositório, que por sua vez mantém uma estrutura globalmente conhecida e que, através de mapeamentos, permite acessar os dados de diversas fontes distribuídas e com estruturas heterogêneas.

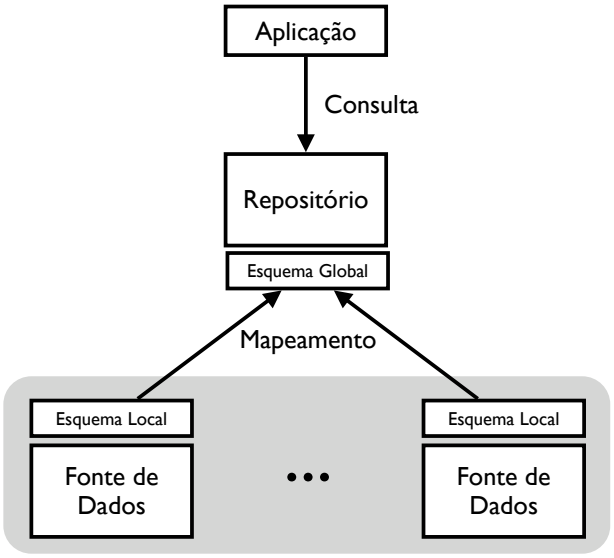
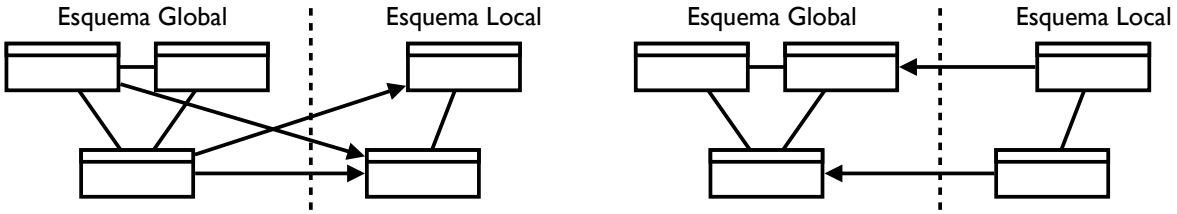


Figura 2.1: Exemplo de arquitetura para um sistema de integração de dados.

Para modelar o esquema intermediário e seus mapeamentos existem duas abordagens clássicas, conforme ilustrado na Figura 2.2. Na abordagem *GaV* (Chawathe et al., 1994; Haas et al., 1997), o conteúdo de cada entidade no esquema global está associado com uma visão sobre o esquema de origem. Nesse modelo, a formação de consultas é mais simples, mas a evolução dos esquemas locais não é facilmente suportada. Por outro lado, a abordagem *LaV* (Duschka e Genesereth, 1997; Levy et al., 1996) permite alterações nos esquemas de origem, sem afetar o esquema global, uma vez que os esquemas locais são definidos como visões sobre o esquema global. No entanto, o processamento de consultas nesse tipo de modelagem pode ser complexo. No trabalho de (Friedman et al., 1999), há uma proposta de combinação entre *GaV* e *LaV*, formalizado em uma abordagem *Global-Local-as-View (GLaV)*.



(a) Abordagem *Global-as-View* (b) Abordagem *Local-as-View*

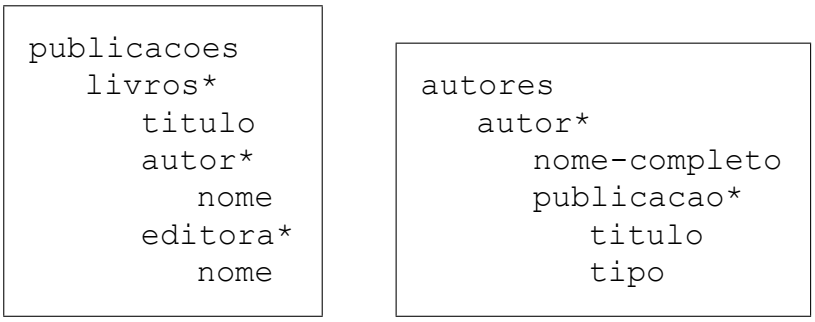
Figura 2.2: Abordagens clássicas para modelagem de sistemas de integração de dados.

2.2 Integração de Dados e XML

Durante anos a literatura focou apenas na integração de dados clássica, ou seja, para dados representados no modelo relacional. No entanto, no início da década passada os pesquisadores voltaram seus interesses para um novo e emergente modelo de dados, o XML (Bray et al., 1998). O novo modelo transformou-se em padrão para troca de dados, tornando-se a escolha ideal para sistemas que procuram interoperabilidade de dados. Assim, o XML e suas linguagens de consulta são as interfaces escolhidas para Serviços Web, banco de dados XML e tantas outras aplicações que envolvem gerenciamento e troca de dados.

Integrar dados de diferentes documentos XML recorre aos mesmos problemas descritos na literatura para a integração clássica. No entanto, novas soluções precisam ser encontradas para tratar as particularidades deste novo cenário.

Em um processo de integração de dados, um dos principais problemas existente é a heterogeneidade das fontes. Para ilustrar esse problema, observe a Figura 2.3 na qual dois esquemas simples, para descrever livros e autores, podem ser representados de formas distintas. Na Figura 2.3(a) tem-se as publicações agrupadas por livros, enquanto que na Figura 2.3(b) as publicações são organizadas por autores.



(a) Esquema da Fonte 1 (b) Esquema da Fonte 2

Figura 2.3: Estruturas distintas para representar o mesmo tipo de informação.

Para alcançar a interoperabilidade completa de dados, todos os problemas causados por sua heterogeneidade precisam ser resolvidos (Cruz e Xiao, 2005). Segundo Bishr (Bishr, 1998), fontes de dados podem ser heterogêneas em sintaxe, esquema ou semântica, tornando a integração de dados uma tarefa bastante complicada. A heterogeneidade

sintática é causada pelo uso de modelos ou linguagens diferentes. Heterogeneidade de esquemas acontece devido a diferenças estruturais na representação dos dados. Por fim, a heterogeneidade semântica é causada pela diferença de significados ou interpretação dos dados em vários contextos.

Com a linguagem XML tornando-se um padrão para troca de dados na internet, criou-se, desta forma, uma plataforma sintática. A criação dessa plataforma auxiliou a resolução dos problemas causados pela heterogeneidade sintática. No entanto, a heterogeneidade de esquemas ainda ocorre, pois depende dos esquemas XML usados. Porém, definindo uma padronização de esquemas essa heterogeneidade pode ser resolvida. Já a heterogeneidade semântica pode persistir mesmo que as heterogeneidades sintática e de esquemas não ocorram, como em casos nos quais duas entidades distintas referem-se ao mesmo conceito no mundo real.

A heterogeneidade semântica aparece em diferentes formas, como por exemplo, quando o mesmo termo é usado para descrever coisas distintas, ou quando o mesmo conceito é representado por termos distintos entre as fontes de dados. A heterogeneidade semântica surge também quando diferentes unidades são utilizadas em duas ou mais fontes para descrever uma propriedade de um conceito global. Como exemplo, imagine o caso no qual duas fontes armazenam o peso de uma pessoa, uma em quilogramas e outra em libras. Ao executar uma consulta solicitando a pessoa mais pesada em quilogramas ou quando uma atualização for realizada no repositório, no qual a medida é dada em libras, uma função de transformação precisa ser aplicada sobre o dado para adequá-lo à unidade correspondente.

A necessidade de expressar o significado dos dados conforme seu entendimento geral, faz com que sistemas de integração de dados implementem algum processo para compartilhar conceitos ou decisões já tomadas sobre determinados conflitos. Baseado nesse conhecimento, o sistema pode resolver, automaticamente, uma grande parcela dos novos conflitos relacionados ao significado dos dados no contexto em que eles estão inseridos. Uma abordagem de limpeza proposta neste trabalho, que será detalhada no capítulo 4, é implementar um mecanismo de política de fusão. Este mecanismo possibilita ao administrador do repositório de dados definir regras de resolução de conflitos, baseado na

experiência adquirida durante as integrações passadas. Isto é, regras que descrevem, através de estratégias, as decisões que seriam comumente tomadas pelo usuário no caso de limpeza manual dos dados.

2.3 Problemas Relacionados

Quando pretende-se manter dados em um repositório integrado, algumas necessidades são recorrentes, tais como manter mapeamentos entre fonte e repositório, identificar entidades duplicadas, manter a proveniência dos dados e principalmente garantir a qualidade dos dados com um processo eficiente de limpeza. Essa seção descreve estes problemas e algumas abordagens propostas na literatura para solucioná-las.

2.3.1 Detecção de Duplicidade

O problema de detecção de duplicidade tem sido estudado em diversas áreas e com muitas denominações, como por exemplo deduplicação, resolução de referências e reconciliação. Este processo é um passo muito importante no processo de integração de dados, que consiste em combinar diferentes representações de um objeto do mundo real em uma representação única.

Para apresentar alguns problemas encontrados na integração de dados e a dificuldade de interpretar corretamente os conflitos identificados, imagine o seguinte cenário: Uma base de dados sobre publicações, por exemplo DBLP (Ley, 2010), pode ter diferentes registros com nomes de autores “R. D. Fley”, “Roberto D. Fley” e “Roberto Daniel Fley” que referem-se a mesma pessoa. Na falta de identificadores como Cadastro de Pessoas Físicas (CPF) ou Registro Geral (RG) esta ocorrência pode levar a diferentes problemas, como registros redundantes, estatísticas incorretas, entre outros.

A detecção de duplicidade é um problema difícil e não pode ser resolvido usando apenas casamentos exatos de atributos, pois há o problema de identificação, onde diferentes representações referem-se à mesma entidade. No exemplo anterior, “Roberto D. Fley” e “Roberto Daniel Fley” são a mesma pessoa e uma amostra desse problema. Falhas na

identificação afetam a completude da informação. Por outro lado, é possível que dois registros com o nome "R. D. Fley" e o mesmo endereço refiram-se a dois irmãos e não a mesma pessoa. Este problema pode afetar a precisão e é conhecido como desambiguação.

Segundo Bleiholder e Naumann (Bleiholder e Naumann, 2008), as duas principais dificuldades a serem resolvidas no processo de detecção de duplicidade são a efetividade e a eficiência do processo. A efetividade é, na maioria da vezes, afetada pela qualidade das medidas de similaridades e da escolha de seus limites. Um limite de similaridade determina quando dois objetos são duplicados. Desta forma, um limite muito baixo produzirá um resultado com muitas duplicidades detectadas, mas com baixa precisão. Já um limite muito alto tem alta precisão, mas pode não determinar todas as duplicidades. Portanto, a grande dificuldade é determinar o valor ideal a ser usado como limite.

A eficiência, que procura obter a máxima produtividade com o menor esforço, é um tópico importante na medida que os conjuntos de dados são frequentemente grandes. Assim, calcular e armazenar os pares de objetos torna-se um obstáculo. Para superar este obstáculo define-se um particionamento inteligente dos objetos e comparação dos pares de objetos somente dentro de uma partição.

O resultado do processo de detecção de duplicidade é a atribuição de um identificador de objeto para cada representação. Duas (ou mais) representações com o mesmo identificador indicam duplicidade. Este é o primeiro passo em sistemas de integração de dados. O segundo é solucionar os conflitos ocorridos nos dados destas entidades.

Para o problema específico de resolução de entidades, há uma quantidade considerável de trabalhos disponíveis tratando-se de dados relacionais. Alguns trabalhos, focados no desempenho, baseiam-se em transformações e detecção de duplicidade realizadas pelo usuário (Hernández e Stolfo, 1995; Raman e Hellerstein, 2001). Em diversas áreas, a detecção de duplicidade recebe uma atenção especial e, normalmente, com soluções dependentes do domínio. Por exemplo, dados genealógicos (Quass e Starkey, 2003), o conjunto de dados Census (Winkler, 1994) e dados bibliográficos (Hylton, 1996). Soluções independentes de domínio incluem os trabalhos relatados em (Jin et al., 2003; Yan e

Garcia-Molina, 1995). Todos estes trabalhos tratam da duplicidade de dados no modelo relacional, no qual a estrutura é claramente definida.

Para garantir a unicidade de entidades em dados XML, várias técnicas tem sido propostas na literatura (Fallside, 2000; Pemberton et al., 2002). Para este trabalho a identificação é feita por chaves para XML primeiramente definidas em (Buneman et al., 2001a).

2.3.1.1 Chaves para XML

Para auxiliar no entendimento do modelo descrito neste trabalho, esta seção descreve brevemente o conceito de chaves para XML. Tais chaves serão utilizadas para identificar e representar as entidades do repositório.

Antes de definir as chaves para XML, faz-se necessário a apresentação de **expressão de caminho**, a qual é um subconjunto de XPath e expressões regulares (Hopcroft e Ullman, 1979). É possível seguir o caminho P e depois o caminho Q através da operação de concatenação P/Q. Além disso, os símbolos "ϵ" e "//" representam, respectivamente, um caminho vazio e um caminho arbitrário.

Três partes precisam ser definidas para descrever uma chave XML: 1) o *contexto* no qual a chave deve atuar; 2) um conjunto *destino* no qual a chave está sendo definida; e 3) *valores* que distinguem cada elemento do conjunto destino.

Seguindo a sintaxe de (Buneman et al., 2001a) uma chave XML é escrita como:

$$\varphi : (Q_1, (Q_2, \{P_1, \ldots, P_p\}))$$

onde φ é o identificador da chave, expressões de caminho Q_1 e Q_2 definem o contexto e o destino, respectivamente, e $P_1, ..., P_p$ são os caminhos da chave. Para este trabalho, restringiu-se os caminhos da chave para serem atributos $@A_1, ..., @A_p$ ou elementos com valores textuais. Uma chave é dita *absoluta* se o caminho do contexto Q_1 é um caminho vazio, e *relativa* caso contrário.

2.3.2 Proveniência dos Dados

A proveniência (também conhecida como linhagem) pode ser descrita de diferentes formas, conforme o contexto da aplicação. No contexto de sistemas de banco de dados, Buneman *et al* (Buneman et al., 2001b) definem proveniência de dados como a descrição da origem do dado e o processo pelo qual ele passou até ser armazenado no repositório. Manter informações sobre proveniência constitui a prova de confiança dos dados e, por sua vez, ajuda a determinar a qualidade e confiança que podem ser atribuídas aos dados (Tan, 2007).

Nesta seção são apresentadas as aplicações de proveniência, principalmente com objetivo de auxiliar a garantir a qualidade dos dados. Também são descritas as principais técnicas para anotar as proveniências dos dados. Por fim, são apresentadas as formas de manter o endereçamento dos nós na árvore XML.

2.3.2.1 Aplicação de Proveniência

As formas de proveniência podem ser construídas para dar suporte a uma série de atividades em sistemas de banco de dados. Estas informações podem ser aplicadas em processos como:

Qualidade dos Dados: a proveniência pode ser usada para estimar a qualidade e confiança do dado baseado na fonte de dados e suas transformações.

Trilhas de Auditoria: usar a proveniência em trilhas de auditoria é especialmente útil quando deseja-se traçar o caminho que o dado percorreu, determinar a utilização dos dados e detectar erros na geração de dados.

Atribuição: a proveniência pode estabelecer a propriedade e os direitos dos dados, tornando possível sua citação e determinando a responsabilidade em casos de utilização de dados errôneos.

Informacional: um uso genérico de proveniência é a utilização de consultas baseadas na proveniência dos metadados para realizar a descoberta de dados. Isto também pode ser utilizado para prover um contexto que facilite a interpretação dos dados.

Dentre essas aplicações de proveniência, a que tem maior relevância para o objetivo desta proposta é a aplicação para garantir a qualidade dos dados. O conhecimento sobre a origem do dado auxilia na qualidade da fonte de dados uma vez que dados inconsistentes podem ser introduzidos em uma fonte e propagados para outras fontes derivadas da primeira. Se for disponibilizado um conhecimento semântico da proveniência, é possível automatizar a avaliação da qualidade dos dados baseada em métricas definidas e prover, através do uso de metadados, um indicativo de confiança para as fontes.

2.3.2.2 Técnicas de Proveniência

Buneman *et al* (Buneman et al., 2006) apresentam quatro técnicas de efetuar anotações de proveniência de dados de acordo com o nível de atualização a ser armazenado. São elas: Proveniência Direta, Proveniência Transacional, Proveniência Hierárquica e Proveniência Transacional-Hierárquica.

Proveniência Direta. Este método armazena um registro de proveniência para cada dado copiado, inserido ou apagado. Além disso, cada operação de atualização é tratada como uma transação separada. Esta técnica pode ser dispendiosa em termos de espaço. Entretanto, esse método retém o máximo possível de informação sobre as ações do usuário. De fato, a operação exata de atualização descrevendo a sequência de ações do usuário pode ser recuperada da tabela de proveniência.

Proveniência Transacional. Neste técnica, as atualizações são agrupadas em transações maiores que uma operação isolada e somente ligações de proveniência são armazenadas, descrevendo o conjunto de mudanças resultantes de uma transação. Por exemplo, se o usuário copia dados de uma fonte S_1, apaga-os em seguida, copia dados de outra fonte S_2 no seu lugar, e finaliza a transação, isto tem o mesmo efeito que a simples cópia de dados de S_2. Desta forma, detalhes sobre estados intermediários ou armazenamento de dados temporários na base de dados não são mantidos, porém as anotações ocupam menos espaço de armazenamento.

Proveniência Hierárquica. Em um modelo de dados hierárquico, tanto na proveniência direta como na transacional, grande parte da informação de proveniência tende a ser

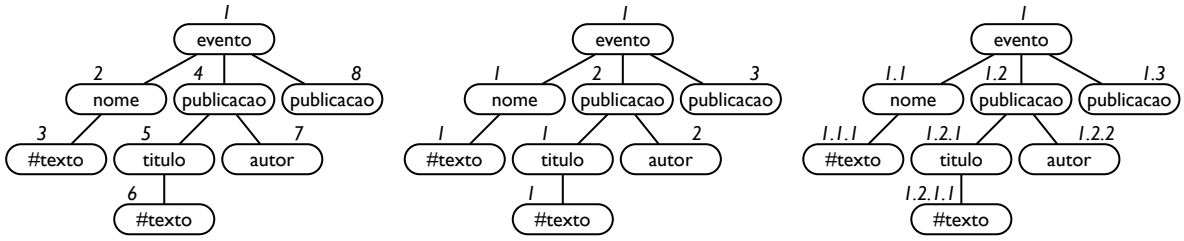
redundante, uma vez que em muitos casos a anotação de um nó filho pode ser inferida da anotação do seu pai. Por esse motivo, pode-se considerar uma outra técnica, chamada de proveniência hierárquica. Sua ideia principal é que não existe necessidade de armazenar todas as ligações de proveniência explicitamente, pois a proveniência de um filho de nó copiado pode ser inferida da proveniência do seu pai. Assim, na proveniência hierárquica são armazenadas somente as ligações de proveniência que não podem ser inferidas.

Proveniência Transacional-Hierárquica. Nesta técnica considera-se a combinação de duas técnicas: transacional e hierárquica. A proveniência transacional-hierárquica pode ser muito mais clara e objetiva que qualquer outra abordagem individualmente.

2.3.2.3 Endereçamento de Nós em Árvore XML

Manter o endereçamento dos nós é uma tarefa importante para o processo de resolução de conflitos dos dados quando deseja-se manter um histórico de resoluções e, posteriormente, utilizar esse histórico como auxílio na resolução de novos conflitos envolvendo a mesma entidade, isto é, a entidade que tem o mesmo endereço em determinada fonte.

Em (Tatarinov et al., 2002) são analisadas três formas de endereçamento de nós em árvores XML que permitem manter a ordem dos elementos dentro do documento quando for necessário efetuar alguma transformação. Conforme mostra a Figura 2.4, as três formas de endereçamento são: Ordem Global, Ordem Local e Ordem de Dewey.



(a) Ordem Global (b) Ordem Local (c) Ordem de Dewey

Figura 2.4: Ilustração dos métodos para codificação de endereçamento dos nós.

Ordem Global. Com a Ordem Global, a cada nó é atribuído um número que representa posição absoluta no documento. Qualquer esquema de numeração pode ser usada

desde que seja consistente com a ordem do documento. A Figura 2.4(a) apresenta um documento com nós identificados de acordo com a Ordem Global.

Ordem Local. Nesta forma cada nó recebe um número que representa sua posição relativa entre seus irmãos, como ilustrado na Figura 2.4(b). A Ordem Local é suficiente para recriar a ordem do documento, uma vez que a combinação da posição de um nó com aquela do seu ancestral forma um vetor de caminhos que identifica unicamente a posição absoluta do nó dentro do documento. Em outras palavras, esse vetor de caminhos provê um ordenamento global dos nós.

Ordem de Dewey. É baseada na Classificação Decimal de Dewey desenvolvida para a classificação genérica de conhecimento (Chan e Mitchell, 2003). Com a Ordem de Dewey, a cada nó é atribuído um vetor que representa o caminho da raiz do documento até o nó. Cada componente do caminho representa a ordem local de um nó ancestral, como ilustrado na Figura 2.4(c). Cada caminho identifica unicamente a posição absoluta do nó dentro do documento.

Foi mostrada nessa seção a utilidade que as anotações sobre proveniência adicionam aos repositórios de dados integrados. Através dessas informações pode-se tornar o processo de limpeza de dados mais eficiente e confiável, melhorando a qualidade dos dados armazenados. Além disso, essas informações são fundamentais para modelos de integração nos quais deseja-se permitir a reconstrução da fonte de dados que originou a atualização no repositório.

2.3.3 Limpeza de Dados

A limpeza de dados consiste no processo de detectar e remover erros e inconsistências dos dados com a finalidade de melhorar a qualidade dos mesmos (Rahm e Do, 2000). Quando múltiplas fontes de dados precisam ser integradas em um repositório central, a necessidade de limpeza e seleção dos dados cresce significativamente. Isto ocorre principalmente porque as fontes contêm dados representados de forma diferente.

Os problemas com a qualidade dos dados existem tanto no nível de esquema como no nível de instância. No primeiro caso, os problemas são decorrentes da falta de um modelo

adequado ou da aplicação de restrições de integridade específicas, como por exemplo as limitações do modelo de dados ou a má concepção do esquema. Os problemas no nível de instância dizem respeito aos erros e inconsistências que não podem ser prevenidos no nível de esquema como, por exemplo, erros de escrita, falta de valores, referências incorretas, entre outros.

Um resumo dos problemas no nível de esquema e de instância pode ser visto nas Tabelas 2.1 e 2.2, respectivamente. No entanto, para o desenvolvimento deste trabalho serão considerados somente os problemas de qualidade dos dados no nível de instância.

Tabela 2.1: Problemas no nível de esquema (adaptado de (Rahm e Do, 2000)).

Escopo	Problema	Inconsistência	Motivo
Atributo	Valores ilegais	dataNasc=30.13.70	Valores fora dos limites aceitáveis
Registro	Dependência de atributo	idade=22,dataNasc=12.02.70	A idade deveria ser data atual menos data de nascimento
Tipo de Registro	Violação de unicidade	emp_1=(nome="John Smith", SSN="123" emp_2=(nome="Peter Miller", SSN="123"	Violação de unicidade para o atributo SSN
Fonte	Violação de integridade referencial	emp=(nome="John Smith", deptno=127	Departamento referenciado (127) não está definido na fonte

Tabela 2.2: Problemas no nível de instância (adaptado de (Rahm e Do, 2000)).

Escopo	Problema	Inconsistência	Motivo
Atributo	Abreviações	ocupacao="Programador Jr."	Valores iguais descritos de forma distinta
	Valores embutidos	nome="J. Smith 12.02.79 N.Y."	Múltiplos valores em um atributo
Registro	Violação de dependência	cidade="Redmond", cep=77777	Cidade e CEP deveriam ser correspondentes
Tipo de Registro	Registros contraditórios	emp_1=(nome="J. Smith", dataNasc=12.02.70 emp_2=(nome="J. Smith", dataNasc=12.12.70	A mesma entidade descrita de forma diferente
Fonte	Referências incorretas	emp=(nome="John Smith", deptno=17	Departamento referenciado (17) está definido, mas incorretamente

Segundo (Rahm e Do, 2000), em geral, o processo de limpeza de dados envolve diversas etapas e um breve resumo de cada etapa é apresentado a seguir:

Análise dos dados: uma análise detalhada é necessária devido a necessidade de detectar quais tipos de erros e inconsistências devem ser removidos. Além de uma inspeção manual dos dados ou suas amostras, algoritmos de análise podem ser usados com o objetivo de adquirir informações sobre as propriedades dos dados e detectar possíveis problemas de qualidade.

Definição do fluxo de transformação: um grande número de transformações e etapas de limpeza devem ser executadas dependendo do número de fontes, sua heterogeneidade e seu grau de consistência. Assim, faz-se necessária a definição de regras de mapeamento e transformação. Estas definições devem, sempre que possível, ser definidas através de consultas declarativas e/ou linguagens de mapeamento.

Verificação: a exatidão e a efetividade de um fluxo de transformação e as definições de transformação devem ser testadas e avaliadas como, por exemplo, em uma amostra de dados, para melhorar as definições. Múltiplas iterações das etapas de análise, definição e verificação podem ser necessárias, dado que alguns erros surgem apenas após a aplicação de alguma transformação.

Transformação: esta etapa é uma das mais importantes do processo de limpeza. Este processo de transformação de dados consiste de múltiplos passos, onde cada passo realiza uma transformação relacionada ao esquema ou às instâncias com o objetivo único de mapear as fontes de dados conforme um esquema global. A transformação correta dos dados, uma vez encontrada, pode ser executada através do fluxo normal de extração, transformação e carga dos dados no repositório central. Após esta etapa, todos os objetos de um determinado tipo são representados homogeneamente.

Fluxo de retorno do dado limpo: após a remoção dos erros, os dados limpos devem também ser refletidos nas fontes de origem dos dados com a finalidade de evitar, futuramente, a repetição do trabalho de limpeza em novas extrações de dados.

Tipicamente, o processo de limpeza de dados não pode ser executado sem o envolvimento de um perito do domínio, uma vez que a detecção e correção de anomalias requer conhecimento especializado. Como tal, este processo é por natureza semiautomático, devendo ser o mais automatizado possível em virtude do volume de dados processados e do tempo necessário para que um especialista execute a limpeza manualmente.

Quando sistemas que auxiliam na limpeza de dados são desenvolvidos, diversos requisitos devem ser satisfeitos. O mais importante consiste na detecção e remoção de todas as inconsistências. O sistema deve priorizar a minimização da análise manual e procurar ser extensível de forma a integrar facilmente novas fontes de dados.

2.3.4 Estratégias para Resolução de Conflitos

Estratégias para resolução de conflitos são métodos aplicados em um nível mais alto de tomada de decisão com a finalidade de definir o que fazer com os dados ditos inconsistentes. Algumas dessas estratégias descrevem a decisão sobre qual valor tomar, como combinar os valores, como criar um novo valor ou mesmo solicitar intervenção humana.

Existem diversas estratégias para manipular inconsistências (Fuxman et al., 2005; Motro e Anokhin, 2004; Schallehn et al., 2004). Segundo Bleiholder e Naumann (Bleiholder e Naumann, 2006) elas podem ser classificadas conforme mostrado na Figura 2.5. Devido ao modo como manipulam (ou não) os dados conflitantes, as estratégias foram divididas em três classes: ignorar, anular e resolver conflitos. Na sequência são apresentados detalhes sobre as estratégias que compõem essas classes. Com a finalidade de facilitar a compreensão, alguns nomes de estratégias definidos em inglês tiveram sua adequação para o português.

Figura 2.5: Classificação de estratégias para resolução de conflitos (Bleiholder e Naumann, 2006).

Ignorar conflito. Neste tipo de estratégia não são tomadas decisões com respeito aos conflitos em geral. Quando esta estratégia é aplicada, o sistema não precisa solucionar o conflito no dado, pois a informação ainda não é necessária ou usada. Estas estratégias são fáceis de implementar e também são frequentemente aplicadas em diversas situações de integração. São dois os exemplos desse tipo de estratégia: *Passe adiante* e *Considere tudo*.

- *Passe Adiante (em inglês, Pass It On).* Esta estratégia simplesmente reúne todos os valores conflitantes e passa-os para o usuário ou outra aplicação para que estes decidam como resolver o conflito.

- *Considere tudo (em inglês, Consider All).* Esta estratégia cria possibilidades de valores a partir dos valores conflitantes. Feito isso, apresenta um lista enumerada para que usuário escolha uma solução dentre todas as apresentadas.

Anular conflito. Esta estratégia não resolve realmente o conflito, todavia manipula os dados inconsistentes. Ela não espera que o conflito aconteça para resolvê-lo, ou seja, não considera os valores dos dados antes de tomar a decisão. São estratégias aplicadas quando é necessário uma decisão rápida sobre o dado. Assim, caso haja um conflito em determinado dado sabe-se previamente qual valor tomar. Porém, como as decisões são tomadas antes de conhecer os valores dos dados ou sem buscar todos os valores de dados, a estratégia nem sempre consegue atender conflitos individuais. Desta forma, embora seja um técnica de rápida resolução, existe uma perda de precisão à medida que nem todas as informações disponíveis são levadas em conta na hora da resolução do conflito.

Como mostrado na Figura 2.5, esta classe está subdividida em outras duas, uma que considera os metadados no momento da avaliação do conflito (baseada em metadado) e a outra que não faz essa consideração (baseada em instância). Alguns exemplos dessas subclasses são descritos abaixo.

Duas estratégias baseadas em instância são:

- *Despreze Nulos (em inglês, Take The Information).* Esta estratégia consiste em sempre aceitar a instância com informação, desprezando aquelas que não apresentam informação alguma. Esta estratégia deixa de lado valores nulos e é uma forma natural de trabalhar com incertezas.

- *Sem "Fofocas" (em inglês, No Gossip).* A ideia por trás dessa estrategia é que caso não haja certeza de como solucionar a inconsistência, deixe-a de fora e acate apenas os valores de precisão confirmada.

Um exemplo de anulação de conflito baseado em metadados é aquela que utiliza o nível de confiabilidade da fonte de dados.

- *Confie Nas Amigas (em inglês, Trust Your Friends).* A intenção dessa estratégia é basear-se em uma terceira parte para decidir o valor correto. Uma vez definida a fonte confiável, todos os dados dessa fonte são carregados, independente da existência de conflitos. Intuitivamente a escolha da fonte "amiga" pode ser feita pelo usuário, mas ela pode ser calculada automaticamente, por exemplo, por um sistema de classificação. A escolha pode ser pela mais barata, mais confiável, que tenha maior tamanho ou outro critério de qualidade como no sistema *Fusionplex* (Motro e Anokhin, 2006).

Resolver conflito. Ao contrário das duas classes anteriores, esta estratégia considera todos os dados e metadados antes de resolver o conflito. Além disso, esta classe está subdividida em estratégias decisoras e estratégias mediadoras. A principal característica das decisoras é que elas escolhem seus valores a partir de todos os valores apresentados e podem ou não levar em conta os metadados. As mediadoras, por outro lado, podem escolher um valor que não esteja entre os valores conflitantes, ou seja, pode criar um novo valor que não existia originalmente.

São exemplos de estratégias decisoras baseada em instância:

- *Maria Vai Com as Outras (em inglês, Cry With the Wolves).* Esta estratégia parte do princípio que os valores corretos prevalecem sobre os incorretos. Desta forma, segue-se a maioria, escolhendo o valor que ocorre com mais frequência dentre todos os valores conflitantes.

- *Escolhas às Escuras (em inglês, Roll the Dice).* A partir de todos os valores conflitantes, escolhe-se um aleatoriamente. Quando não se tem ideia de qual valor escolher, tomar um aleatoriamente pode ser uma boa escolha. Embora não seja ideal, é computacionalmente barata e mantém a unicidade de entidades.

Como estratégia mediadora tem-se a:

- *Boa Vizinhança (em inglês, Meet in the Middle).* Esta estratégia segue o princípio da harmonia, ou seja, não prefere nenhum valor em relação aos demais. Ao invés disso, cria um valor tão próximo quanto possível de todos os valores. Isso ajuda a minimizar os erros.

Uma estratégia que representa as decisoras baseadas em metadados é:

- *Acate o Mais Recente (em inglês, Keep Up To Date).* Esta estratégia usa o valor mais recente e requer alguma informação adicional com relação ao tempo. Esta informação pode estar disponível nas fontes através de um atributo específico ou pode ser providenciado através da proveniência dos dados.

2.4 Considerações

Neste capítulo foi apresentado o estado da arte com relação a integração de dados. Definiu-se um sistema de integração de dados e os benefícios que essa abordagem oferece. Além disso, foram discutidos temas fortemente relacionados com o trabalho a ser desenvolvido.

Um processo de integração possui dois passos básicos: identificação das entidades e resolução dos conflitos. Na seção 2.3.1 foi mostrada a importância de efetuar corretamente a identificação das entidades relacionadas e os problemas que a má execução dessa tarefa pode ocasionar no restante do processo de integração.

A Seção 2.3.2 trata as formas para manutenção da proveniência dos dados, que é um tema importante quando pretende-se aplicar estratégias de resoluções baseadas em metadados. Além disso, com a proveniência das fontes é possível, através da reconstrução da fonte original, identificar alterações nas fontes que atualizam o repositório.

Para que seja possível extrair dados com eficiência e precisão, um repositório de dados integrado deve oferecer um estágio para realização da limpeza dos dados armazenados, eliminando dados incorretos, redundantes ou inconsistentes. Assim, a Seção 2.3.3 apresentou alguns problemas que podem surgir quando as informações são mantidas de forma integrada.

Por fim, na seção 2.3.4 foram apresentadas as estratégias para resolução de conflitos que já são utilizadas em bancos de dados relacionais. As estratégias são parte importante desta dissertação, pois é através da combinação destas estratégias que são formadas as regras que compõe a política de fusão proposta nesta dissertação.

Capítulo

3

Trabalhos Correlatos

Com o objetivo de conhecer e entender os requisitos para um sistema de integração e limpeza de dados XML, foram analisados diversos sistemas que utilizam estratégias como forma de resolução de conflitos em nível de instância. Os sistemas foram analisados de acordo com sete critérios e foram avaliados tanto sistemas nos quais o modelo de dados é relacional como os sistemas para integração de dados XML.

3.1 Critérios de Avaliação

Os critérios de avaliação discutidos a seguir procuram descrever as principais características dos sistemas de integração de dados. São apresentadas questões as quais deseja-se responder com a análise destes critérios.

Modelo de Dados. Qual é o tipo de modelo de dados usado internamente ou, caso seja um sistema mediador, qual o modelo do mediador? Quais modelos são permitidos como fonte de dados? São baseados em dados relacionais, ou semiestruturados como XML e OEM, ou ainda em um modelo orientado a objetos?

Modelo de Integração. Qual é a forma de modelagem (*Gav, LaV, GLaV*) adotada pelo sistema? Conforme discutido na Seção 2.1, o modelo de integração *Global-as-View* utiliza um esquema global (usado pelo repositório) como visões do esquema local (utilizado pelas fontes). Já o modelo *Local-as-View* expressa os esquemas locais como visões do

esquema global. A combinação dessas abordagens é a utilização do modelo *Global-Local-as-View* como tentativa de aproveitar os benefícios dos dois modelos anteriores.

Materialização. Esta característica serve para indicar a forma que o sistema trabalha com os dados das fontes. O sistema materializa e armazena fisicamente os dados das fontes integradas ou apenas provê o acesso virtual à essas fontes?

Mapeamento. Procura entender como é feito o relacionamento entre fonte de dados e repositório. Como são resolvidos os conflitos entre o esquema da fonte e o esquema do repositório? Como é feito o casamento de esquemas?

Detecção de Duplicidade. Como o sistema detecta e manipula os objetos duplicados? Este critério descreve como são identificados os objetos que devem ser fusionados em um único objeto do mundo real. Qual é a principal estratégia para realizar a identificação desses objetos?

Estratégias de Resolução de Conflitos. Uma vez conhecida a lista de possíveis estratégias (apresentadas na Seção 2.3.4), quais realmente são implementadas pelo sistema? Ou quais estratégias tem possibilidade de serem usadas pelo sistema?

Especificação da Fusão. Como é definido o processo de limpeza dos dados? Com este critério, procurou-se observar como os sistemas de integração especificam sua forma de fusão dos dados. Se este processo é realizado manualmente ou utiliza-se algum recurso para torná-lo automatizado.

3.2 Sistemas de Integração

Nesta seção são descritos alguns sistemas que durante os estudos apresentaram mais afinidades com o modelo de sistema proposto nesta dissertação. Todos os sistemas existem ao menos como um protótipo de pesquisa, embora somente os sistemas TSIMMIS e Potter's Wheel (Raman e Hellerstein, 2001) estejam disponíveis para *download* e uso na Web.

3.2.1 Fusionplex

A família de sistemas "plex" consiste de três sistemas: Multiplex (Motro, 1999), Autoplex (Berlin e Motro, 2001) e Fusionplex (Motro e Anokhin, 2006). Uma visão geral dos três sistemas está disponível em (Motro et al., 2004). Dentre os sistemas da família "plex", o Fusionplex é o que apresenta maior número de características afins aos demais sistemas apresentados nesta seção e também em relação aos critérios considerados.

Fusionplex é um sistema para integração de múltiplas fontes de informações autônomas e heterogêneas. Ele utiliza estratégias para resolver inconsistências reais entre as fontes de dados. Fusionplex oferece reconhecimento de inconsistências e facilidades de reconciliação baseadas nas informações de qualidade disponibilizadas pela fonte de dados em atributos de anotação ou metadados. O sistema utiliza internamente o modelo relacional, mas é capaz de integrar dados em outros formatos, desde que as fontes sejam capazes de exportar seus dados no formato tabular.

Os esquemas das fontes integradas são considerados heterogêneos, mas sem conter erros, o que significa que o mapeamento entre as fontes de dados e o esquema global pode ser encontrado e representado com a abordagem *Global-Local-as-View*.

O Fusionplex permite que o usuário resolva conflitos de dados em nível de tupla. O sistema agrupa tuplas que representam o mesmo objeto de acordo com uma chave global. Então, utiliza-se de metadados com alta qualidade, armazenados em colunas adicionais, para escolher as tuplas a serem usadas no processo de fusão. Além disso, o sistema aplica funções de fusão `(min, max, any, ...)` nos atributos de cada objeto para apresentá-los como um único valor para o atributo na representação final do objeto.

3.2.2 HumMer

HumMer é a abreviação para Humboldt-Merger, um sistema de integração de dados que permite a integração virtual e semiautomática de fontes de dados remotas e heterogêneas (Bilke et al., 2005).

O sistema é utilizado via uma interface a qual funciona como um assistente e permite detectar semiautomaticamente os mapeamentos entre fontes e o esquema global, mas também permite intervenção do usuário, caso necessário. Definidos os mapeamentos, o sistema também detecta semiautomaticamente as entidades duplicadas e as integra em uma representação limpa. Em cada etapa o usuário pode interferir, modificando o mapeamento proposto, classificando as entidades duplicadas ou declará-las não duplicadas e por fim, definir como os conflitos devem ser solucionados a fim de manter uma única representação da entidade.

Os dados manipulados pelo HumMer são representados no modelo relacional. HumMer pode ser usado como um sistema orientado a consultas, no qual uma consulta é formulada usando uma extensão de SQL e mapeamento de esquemas, e onde a detecção de duplicidade é feita previamente através de configurações padrões.

A forma de resolução de conflitos implementada pelo sistema HumMer é a agregação definida pelo usuário. Esta agregação utiliza funções como `MIN`, `MAX` e `AVG` e estratégias como *Most Recent* – que opta pelo dado mais atualizado, *Coalesce* – que escolhe a informação em relação a valores nulos, e *Choose* – que permite definir uma fonte de confiança.

3.2.3 Potter's Wheel

É um sistema interativo para limpeza de dados que opera sobre somente uma fonte (Raman e Hellerstein, 2001). O sistema está disponível gratuitamente e oferece uma interface gráfica para dados relacionais apenas.

O foco principal está na definição interativa de operações de limpeza realizadas pelo usuário. Um processo de identificação de conflitos executa em segundo plano e apresenta os valores para o usuário, ou colunas completas que supostamente tenham dados "sujos", por exemplo os dados que estão fora dos limites estabelecidos pelo domínio. Então, o usuário do Potter's Wheel define transformações sobre os dados usando como dica as incompatibilidades já detectadas pelo sistema. Desta forma, valores em colunas podem ser reformatados, modificados, copiados para outras colunas ou divididos em duas colunas

e gradativamente construindo transformações melhores para solucionar as inconsistências, conforme ilustra a Figura 3.1.

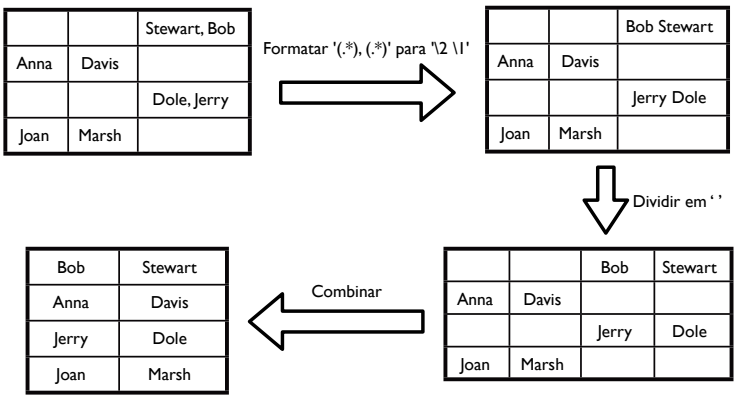


Figura 3.1: Exemplo de aplicação de transformações via sistema Potter's Wheel.

O sistema permite salvar combinações de transformações complexas para que estas sejam executadas em todo o conjunto de dados e não apenas na amostra que é disponibilizada pela ferramenta. Os autores ainda mencionam como uma extensão do sistema as traduções das transformações em procedimentos SQL e XQuery, para que sejam usadas junto com as facilidades proporcionadas pelas otimizações de consultas já existentes.

3.2.4 AJAX

O AJAX é um *framework* na qual a lógica de um programa de limpeza de dados é modelado como um grafo direcionado de transformações que iniciam a partir de uma fonte de dados de entrada e terminam retornando dados limpos (Galhardas, 2006; Galhardas et al., 2000, 2001). Seu principal objetivo consiste em transformar dados de uma ou várias fontes em um determinado esquema destino, tratando uma série de problemas típicos de qualidade dos dados durante o processo.

Uma linguagem declarativa e extensível, baseada em declarações SQL devidamente enriquecidas com um conjunto de primitivas de transformação, possibilita a especificação das transformações de dados (programas de limpeza de dados) de uma forma compacta e de manutenção simplificada. No AJAX encontram-se definidas cinco transformações: **visões SQL** (*SQL view*) – equivale a uma consulta SQL típica, permitindo especificar

uniões e junções SQL; **mapeamento** (*mapping*) – uniformiza o formato dos dados, por exemplo datas, ou simplesmente fusiona ou divide atributos de modo a colocá-los em um formato mais adequado; **detecção de duplicidade** (*matching*) – determina pares de tuplas que, com grande probabilidade, referem-se à mesma entidade; **segmentação** (*clustering*) – com base nos resultados da transformação anterior, agrupa os pares de tuplas que possuem um elevado grau de semelhança, com base em um determinado critério de agrupamento; e **fusão** (merging) – aplicada sobre cada segmento de tuplas com o objetivo de eliminar duplicados, naquilo que é chamada de operação de consolidação.

A semântica de cada uma destas transformações envolve a geração de exceções sobre erros ou inconsistências que possam ocorrer. Estas exceções são o alicerce de um ambiente interativo com o usuário, na manipulação dessas situações. O ambiente também permite a análise dos resultados intermediários, durante a execução de um processo de limpeza de dados. Isto é, o *framework* permite ao usuário, por exemplo, verificar (e alterar), os casamentos realizados antes de iniciar a próxima etapa do processo.

O último aspecto relevante consiste na existência de um mecanismo genealógico dos dados, o que possibilita a obtenção de explicações. Ou seja, para cada transformação de dados, o usuário pode obter informação sobre quais as tuplas que estiveram na base da geração de um determinado registro, algo como uma trilha dos dados.

3.2.5 TSIMMIS

TSIMMIS é um acrônimo para *The Stanford-IBM Manager of Multiple Information Sources*. O objetivo do TSIMMIS é facilitar a integração rápida de fonte de dados heterogêneas que podem incluir tanto dados estruturados como semiestruturados. O sistema usa mediadores para extrair dados das fontes e os converte em um modelo próprio OEM (*Object Exchange Model* (Hammer et al., 1997). OEM é um modelo de dados semiestruturado no qual o dado é representado por uma quádrupla (objetoID, rótulo, tipo, valor) e não há esquema definido como no modelo relacional.

Um mediador converte as fontes de dados em fontes dentro de um modelo central. Os mediadores são especificados utilizando uma linguagem especial, WSL (*Wrapper Specifi-*

cation Language), tanto por especialistas como automaticamente através de métodos de aprendizagem de máquina. Neste sistema, a processamento de consultas segue a abordagem *Global-as-View*, com os mediadores executando o desdobramento das consultas e mostrando o resultado de acordo com a especificação do mediador.

Para identificar objetos, o sistema usa uma função de Skolem que atribui um identificador único e global para objetos do mundo real. A existência de diferentes representações do mesmo objeto em diferentes fontes de dados é detectada, bem como a ocorrência de conflitos entre elas. No entanto, o sistema não permite especificar a forma de resolução de conflitos nos mediadores, embora evite-os usando uma simplificação da estratégia *Trust Your Friends* que permite especificar a fonte preferida na configuração do mediador. Assim, o valor da fonte preferida é utilizado na representação final do objeto.

3.2.6 Nimble

O Nimble é um sistema de integração comercial (Draper et al., 2001). Ele é um dos primeiros sistemas de integração de dados propostos para o modelo de dados XML. Desde seu desenvolvimento, o Nimble vem sendo usado por muitas companhias para limpeza e integração de dados XML provenientes de fontes heterogêneas.

Como este é sistema comercial, são poucos os detalhes e técnicas publicadas. No entanto, a detecção de duplicidade ocorre e o sistema tenta encontrá-las através da mineração de colunas em diferentes tabelas que podem ser agrupadas. Ele assume a existência de pares de valores nas colunas que identificam os mesmos objetos do mundo real. Os índices de agrupamento são construídos semiautomaticamente através de técnicas de mineração de dados e as exceções são manualmente tratas por um usuário especialista.

Não há informações específicas de como os conflitos são solucionados no sistema, mas algumas indicações apontam que um mecanismo de integração baseado em agrupamentos é utilizado juntamente com resoluções de conflitos manuais. Além disso, os autores do Nimble destacam a arquitetura do sistema, a qual é capaz de executar consultas sob demanda em um repositório virtual e também de manter dados materializados localmente.

3.2.7 XClean

O XClean é um sistema para limpeza de dados no modelo XML que permite ao usuário especificar o processo de limpeza de forma declarativa e modular (Weis e Manolescu, 2007). A arquitetura do sistema é formada por três componentes principais: biblioteca de funções, linguagem de programação XClean e compilação e execução através do XQuery (Boag et al., 2007).

A biblioteca de funções oferece ao usuário funções comumente usadas no tratamento de dados, por exemplo, formatadores de data, operações com texto e medidas de similaridade. A linguagem de programação XClean foi desenvolvida para reduzir o esforço do usuário. Esta linguagem provê uma sintaxe específica para operadores de limpeza. Os programas XClean são compilados dentro da linguagem padrão XQuery, o que permite executar seus programas sobre qualquer plataforma XQuery. A execução do programa XClean sobre uma plataforma XQuery resulta em uma fonte XML livre de inconsistências.

Um exemplo de sentença de programa XClean é mostrado na Figura 3.2, a qual ilustra como uma sentença utilizando sintaxe XClean é compilada para uma declaração em XQuery. A ideia por trás das cláusulas XClean é definir comandos de transformações sobre dados "crus" com a finalidade de gerar uma saída limpa e consistente.

Em uma primeira etapa os dados são coletados das diversas fontes e agrupados em uma estrutura de dados "enriquecida". Sobre essa estrutura é executado um processo de detecção de duplicidade utilizando funções de similaridades definidas pelo usuário. Por fim, os objetos são filtrados e, utilizando um operador de fusão, obtém-se uma representação limpa dos dados.

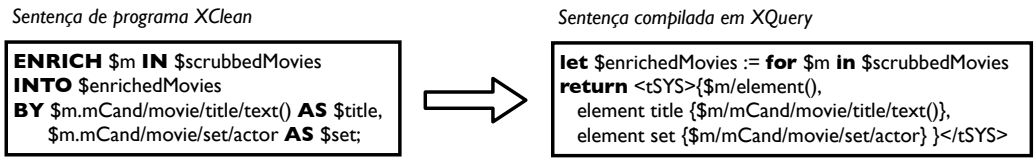


Figura 3.2: Exemplo de sentença XClean convertida em uma declaração XQuery.

3.3 Considerações

Neste capítulo foram apresentados sete sistemas de integração de dados que tem por objetivo solucionar inconsistências nos dados e apresentar um resultado correto e preciso para o usuário final. Os sistemas foram analisados de acordo com critérios de avaliação, relacionando as características comuns em sistemas de integração e que também são aplicáveis ao modelo descrito nesta dissertação.

A Tabela 3.1 apresenta um comparativo entre as principais propriedades dos sistemas de integração de dados apresentados na seção anterior. Quatro utilizam o modelo relacional, um utiliza um modelo de dados próprio (OEM) e dois sistemas específicos para dados XML. Quanto ao modelo de integração, nos sistemas em que esta propriedade é aplicável, HumMer e TSIMMIS utilizam a abordagem *Global-as-View*, na qual o esquema é definido como uma visão sobre os esquemas na fonte original. Nos sistemas AJAX e XClean, que oferecem uma proposta de linguagem declarativa, além da ferramenta Potter's Wheel, que oferece uma interface para execução de interações, não são aplicáveis as abordagens para modelo de integração, pois nos três sistemas o esquema do resultado final depende do conjunto de transformações executadas durante o processo de integração oferecido por cada sistema.

Tabela 3.1: Comparativo entre as propriedades dos sistemas de integração de dados.

Sistema	Modelo de Dados	Modelo de Integração	Materialização	Mapeamento	Detecção de Duplicidade
Fusionplex	Relacional	GLaV	Não	Manual	Assume ID global
HumMer	Relacional	GaV	Não	Semiautomático	ID global gerado por algoritmo próprio
Potter`s Wheel	Relacional	n/a	Sim	n/a	n/a
AJAX	Relacional	n/a	Sim	Manual	ID global gerado por funções de similaridade
TSIMMIS	OEM	GaV	Não	Mediadores	Assume ID global
Nimble	XML	Desconhecido	Desconhecido	Desconhecido	Tabela de mapeamento
XClean	XML	n/a	Sim	Manual	ID global gerado por funções de similaridade

Dos sete sistemas estudados somente Potter's Wheel, AJAX e XClean permitem armazenar o resultado da integração fisicamente. Além disso, também fazem anotações

de proveniência para permitir ao usuário conhecer as etapas de transformações aplicadas sobre os dados.

A realização do mapeamento entre as fontes é realizado na maioria dos sistemas por um usuário especialista, seja na configuração dos mediadores ou declarativamente através da linguagem oferecida por cada sistema. A detecção de duplicidade é outra tarefa que apresenta métodos semelhantes entre os sistemas estudados. Em sua maioria os sistemas utilizam a atribuição de um identificador global para cada entidade representada. Posteriormente, aplicam medidas de similaridade para refinar o resultado e identificar com maior precisão as representações duplicadas.

Na Tabela 3.2, são apresentadas as principais estratégias aplicadas na resolução de conflitos em cada sistema e como essas estratégias são especificadas e aplicadas para cada processo de fusão executado.

Os sistemas Fusionplex e HumMer destacam-se por aplicarem eficientemente as estratégias da classe de resolução de conflitos como *Keep up to date* e *Meet in the middle*. Por outro lado, os sistemas AJAX e XClean oferecem um fluxo de transformações bem definido e permitem declarativamente construir processos de limpeza que possam ser reutilizados, isto é, permitem reaplicar as mesmas técnicas de limpeza em integrações posteriores de forma semiautomática.

Tabela 3.2: Comparativo entre estratégias e forma de fusão dos sistemas de integração.

Sistema	Estratégias de Resolução	Especificação da Fusão
Fusionplex	*Keep to up date*	Manualmente, via consulta
HumMer	*Keep up to date, Trust your friends, Meet in the middle, Pass it on, Roll the dice, ...*	Manualmente, via consulta
Potter`s Wheel	*Pass it on*	Manualmente, via transformações
AJAX	*Keep up to date, Trust your friends, Meet in the middle* e outras definidas pelo usuário	Semiautomaticamente, via fluxo de transformações
TSIMMIS	*Trust your friends*	Manualmente, via regras no mediador
Nimble	*Pass it on*	Manualmente
XClean	Funções definidas ou acopladas pelo usuário	Semiautomaticamente, via programação XClean

Sistemas como o Nimble e Potter's Wheel apenas demonstram o conflito e oferecem ferramentas para que o usuário tome as decisões manualmente. Enquanto o TSIMMIS, único sistema analisado que faz uso de mediadores, utiliza uma única estratégia (*Trust*

your friends) como forma de evitar os conflitos e faz isso permitindo que o usuário configure o mediador para executar a limpeza durante o processo de consulta.

De todos os sistemas pesquisados apenas dois deles apresentam suporte para integração de dados XML. Baseado nisso e também no crescimento acelerado do uso deste modelo de dados, este trabalho assumiu como desafio criar um sistema de integração de dados para XML que permita materializar os dados em um repositório central e oferecer um instrumento para resolução semiautomática das inconsistências encontradas no processo de integração. O modelo proposto é discutido no Capítulo 4 e em seguida, no Capítulo 5, é apresentada a ferramenta XFusion que foi desenvolvida para validar as funcionalidades implementadas no sistema.

Capítulo

4

Modelo de Limpeza em Dados XML

Neste trabalho é proposto um modelo para limpeza de dados durante o processo de integração de dados no formato XML, utilizando um repositório centralizado. Entende-se que o repositório armazena dados de diversas fontes com o intuito de fornecer informações precisas às instituições ou organizações que se valem dos benefícios deste tipo de solução. O trabalho tem como foco reduzir a demanda manual do usuário na fase de limpeza dos dados, sem causar prejuízos na qualidade das informações armazenadas. Para isso, este modelo sugere a utilização de uma política de fusão que, definida pelo usuário, indica como os eventuais conflitos devem ser solucionados em determinados contextos do repositório.

4.1 Cenário de Integração de Dados

Esta seção tem como objetivo descrever o cenário que motivou o desenvolvimento de um modelo para resolução de conflitos entre instâncias provenientes de fontes heterogêneas e que integram um repositório central de dados.

Geralmente o processo de limpeza dos dados necessita da interação humana para garantir a qualidade dos dados. Partindo deste princípio e com a intenção de diminuir o esforço realizado pelo usuário, esta dissertação sugere um modelo de integração de dados no qual o usuário pode definir uma política de fusão baseada em decisões que são comumente tomadas. Assim, sempre que houver novas integrações, a política é aplicada

com a finalidade de solucionar automaticamente os conflitos nos quais o usuário precisaria intervir manualmente.

A Figura 4.1 ilustra o cenário inicial de integração de dados considerado neste trabalho. Neste cenário, diversas fontes de dados estão disponíveis para integração em um repositório central de dados. Um processo de integração, sugerido em (do Nascimento e Hara, 2008), identifica e representa no repositório as inconsistências entre instâncias de fontes distintas. Neste momento, o usuário executa uma série de decisões sobre os conflitos para determinar os valores corretos. Estas decisões são traduzidas em regras de resolução com a finalidade de compor uma base de decisões ou política de fusão, como foi chamada nesta dissertação.

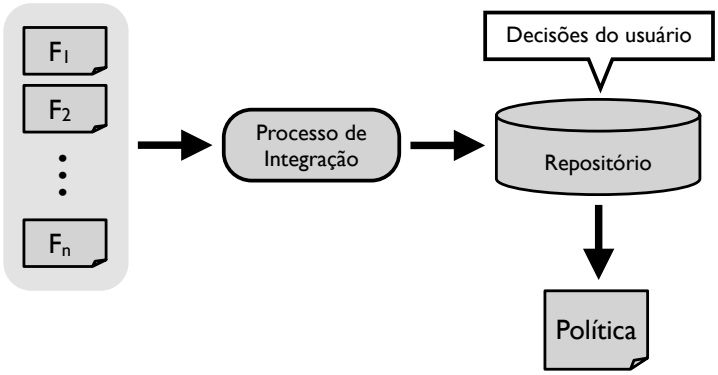


Figura 4.1: Processo de integração de dados em um repositório central.

A política é uma forma do usuário governar entidades do repositório através da composição de regras de resoluções que permitem executar estratégias comumente utilizadas nas decisões tomadas manualmente. A Figura 4.2 ilustra como fica o cenário completo de integração com a utilização do modelo proposto nesta dissertação. O usuário, ao resolver manualmente os conflitos surgidos em uma primeira integração, define um conjunto de regras para solucionar conflitos reincidentes. Desta forma, quando novas versões das fontes de dados forem integrar o repositório, as inconsistências detectadas pelo processo de integração serão passadas a um novo estágio, no qual haverá a aplicação da política com a finalidade de solucionar automaticamente os conflitos, propagando somente os valores corretos ao repositório. Adicionalmente, os dados desconsiderados são arquivados em um histórico para futuras reavaliações e também para permitir a reconstrução da fonte, processos que serão discutidos ao longo desse capítulo.

Como consequência da aplicação da política, o usuário apenas necessitará solucionar inconsistências em atributos que não estão cobertos por nenhuma regra. Definida uma regra para estes atributos, nas integrações subsequentes as inconsistências reincidentes são resolvidas automaticamente. Assim, cria-se um círculo de evolução da política, o qual implica cobrir a maior parte do repositório com as regras de resoluções mais adequadas, reduzindo o esforço manual despendido pelo usuário no processo de limpeza dos dados.

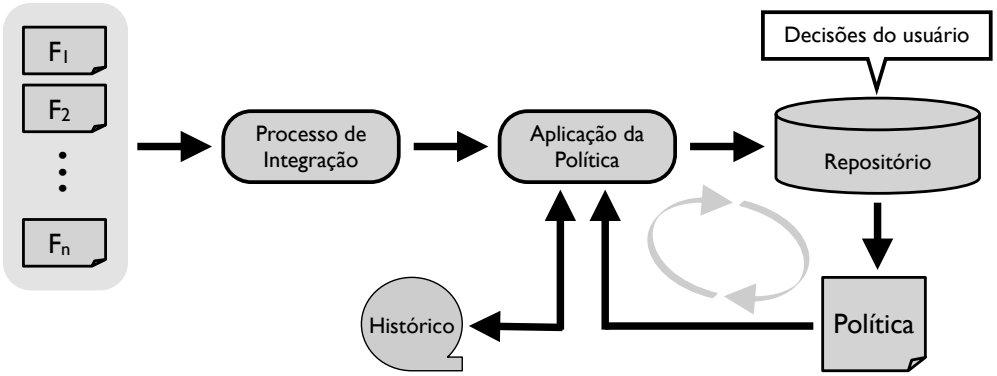


Figura 4.2: Processo de integração de dados com aplicação da política de fusão.

4.1.1 Exemplo Corrente

Nesta seção são apresentados os documentos XML das fontes de dados utilizadas ao longo deste trabalho para caracterizar as funcionalidades do modelo proposto. Adicionalmente, são feitas algumas considerações sobre a estrutura do repositório e a relação com as fontes de dados no momento da integração.

Considere os fragmentos de três fontes de dados distintas F_1, F_2 e F_3, conforme mostra a Figura 4.3. Estas fontes representam dados de produtos eletrônicos informados pelas lojas revendedoras e também pelos fabricantes dos produtos. As Fontes F_1 e F_3 são provenientes de duas lojas distintas, `'Fast'` e `'Extra'` respectivamente, enquanto que F_2 é uma fonte proveniente do fabricante de produtos `'HP'`. Pode ser observado que a fonte proveniente do fabricante agrupa seus produtos por categoria, enquanto que as fontes provenientes de lojas descrevem apenas uma coleção de itens, sem nenhum agrupamento.

O intuito do repositório é manter cotações de diversos produtos para facilitar a consulta e comparação de preços entre os produtos. Então, no momento de integrar essas fontes

Fonte F_1

```
<loja>
  <nome>Fast</nome>
  <item serial="007">
    <fabricante>HP</fabricante>
    <modelo>dv6000</modelo>
    <cor>preto</cor>
    <preco>2980,00</preco>
  </item>
  <item>
    ...
  </item>
  ...
</loja>
```

Fonte F_2

```
<empresa>
  <nome>HP</nome>
  <categoria tipo="notebook">
    <produto>
      <modelo>dv6000</modelo>
      <cor>branco</cor>
      <origem>Brasil</origem>
    <produto>
    <produto>
    ...
    </produto>
  </categoria>
  ...
</empresa>
```

Fonte F_3

```
<loja>
  <nome>Extra</nome>
  <item serial="007">
    <fabricante>HP</fabricante>
    <modelo>dv6000</modelo>
    <cor>preto</cor>
    <preco>2600,00</preco>
  </item>
  <item>
    ...
  </item>
  ...
</loja>
```

Figura 4.3: Exemplo de fragmentos de fontes de dados heterogêneas.

definiu-se que um produto é identificado unicamente pela dupla `fabricante` e `modelo`. Além disso, um produto pode ter várias cotações e cada cotação é unicamente identificada pelo elemento `loja`, o qual contém o nome da loja responsável pela informação.

A estrutura do repositório pode ser conhecida pelo mapeamento estabelecido entre fontes e repositório. Isto é, cada fonte que integra o repositório tem um mapeamento definido sobre o esquema virtual do repositório. Assim, cada entidade na fonte é relacionada com seu destino no repositório.

O mapeamento entre a fonte F_2 para o repositório tem a seguinte configuração:

$$
\begin{aligned}
&/produto && \longleftarrow && /empresa/categoria/produto \\
&/produto/fabricante && \longleftarrow && /empresa/nome \\
&/produto/modelo && \longleftarrow && /empresa/categoria/produto/modelo \\
&/produto/cor && \longleftarrow && /empresa/categoria/produto/cor
\end{aligned}
$$

Enquanto que o mapeamento entre as Fontes F_1 e F_3 para o repositório está especificado da seguinte forma:

$$
\begin{aligned}
&/produto && \longleftarrow && /item \\
&/produto/@serial && \longleftarrow && /item/@serial \\
&/produto/fabricante && \longleftarrow && /item/fabricante \\
&/produto/modelo && \longleftarrow && /item/modelo \\
&/produto/cor && \longleftarrow && /item/cor \\
&/produto/cotacao/loja && \longleftarrow && /nome
\end{aligned}
$$

/produto/cotacao/preco ⟵ */item/preco*

Nos mapeamentos apresentados, os caminhos ao lado esquerdo representam a estrutura do repositório, enquanto que ao lado direito são os caminhos na fonte original. Desta forma, sempre que for necessário conhecer o esquema do repositório, é possível inferir sua estrutura a partir dos documentos de mapeamentos existententes.

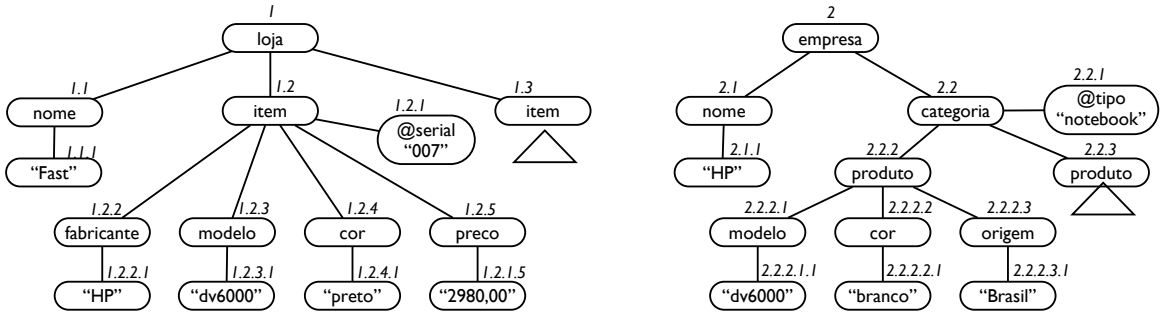
4.2 Representação dos Dados

Baseado no modelo definido em (do Nascimento e Hara, 2008), o repositório de dados integrado é uma árvore XML contendo os dados importados de uma ou mais fontes de dados XML. Todos os níveis da árvore possuem nós identificáveis através de chaves XML, que determinam como identificar unicamente as entidades em uma fonte de dados e permitir a integração de dados provenientes de fontes distintas. O repositório apresenta um esquema fixo, utilizando uma abordagem de visão local, conforme apresentado na Seção 2.1. Embora o esquema não exista fisicamente na forma de documento de esquema, ele pode ser conhecido a partir dos documentos de mapeamento entre repositório e fontes.

Assim como no modelo de Nascimento (do Nascimento e Hara, 2008), o modelo proposto nessa dissertação mantém no repositório informações sobre a proveniência dos dados que integram o repositório. Adicionalmente, é proposto um histórico de resoluções, detalhado na Seção 4.4, no qual são armazenados os dados desconsiderados no momento da integração, isto é, dados da fonte original que foram considerados incorretos por alguma regra de resolução durante o estágio de limpeza dos dados. Com esse registro de histórico é possível reconstruir a fonte original e também reconsiderar valores antigos em novos processos de integração. Além disso, com a separação dos dados incorretos, mantém-se um repositório integrado mais enxuto e com maior precisão no resultado das consultas. Contudo, a principal contribuição deste trabalho ao modelo definido por Nascimento é a inclusão do estágio de limpeza dos dados baseado em uma política de fusão construída pelo usuário. Os detalhes desta política e das regras de resolução que a compõe são descritos na Seção 4.3.

4.2.1 Árvore XML

Para melhor descrever as características do modelo proposto nesta dissertação, faz-se necessário algumas considerações sobre árvore XML. Para ilustrar essa estrutura, observe a Figura 4.4 a qual mostra as representações dos documentos XML que descrevem as fontes F_1 e F_2 no formato de árvore XML.



(a) Árvore XML da Fonte F_1 (b) Fonte F_2: Árvore XML

Figura 4.4: Exemplos de fontes de dados representadas no formato de árvore XML.

Formalmente a representação de um documento com dados XML como uma árvore XML é definida como segue.

Definição 4.1. Dado um conjunto infinito $\mathcal{E}$ de nomes de elemento, um conjunto infinito $\mathcal{A}$ de nomes de atributos e um símbolo $\mathtt{S}$ indicando textos (PCDATA), uma árvore XML $\mathcal{T}$ é definida pela 8-upla $\langle s, \mathcal{V}, r, id, lab, ele, att, val\rangle$, onde

- s é a identificação da fonte de dados;
- $\mathcal{V}$ representa um conjunto de nós em $\mathcal{T}$;
- r representa o nó raiz da árvore;
- id é uma função que atribui identificadores únicos aos nós em $\mathcal{V}$. Dado um nó v, $id(v)$ retorna um vetor que representa o caminho de r até v;
- lab é uma função $\mathcal{V} \rightarrow \mathcal{E} \cup \mathcal{A} \cup \{\mathtt{S}\}$ a qual atribui um rótulo para cada nó em $\mathcal{V}$. Um nó $v \in \mathcal{V}$ é chamado de elemento se $lab(v) \in \mathcal{E}$; se $lab(v) \in \mathcal{A}$, então v é chamado atributo; e chama-se nó texto caso $lab(v) = \mathtt{S}$;

- *ele* é uma função parcial de $\mathcal{V}$ para sequências de vértices em $\mathcal{V}$ tal que para qualquer nó $v \in \mathcal{V}$, se $ele(v)$ é definido então $lab(v) \in \mathcal{E}$;

- *att* é uma função parcial $\mathcal{V} \times \mathcal{A} \rightarrow \mathcal{V}$ tal que para qualquer nó $v \in \mathcal{V}$ e $l \in \mathcal{A}$, se $att(v, l) = v'$ então $lab(v) \in \mathcal{E}$ e $lab(v') = l$;

- *val* é uma função parcial de $\mathcal{V}$ para valores de textos tal que para qualquer nó $v \in \mathcal{V}$, $val(v)$ é um texto se e somente se $lab(v) = \mathtt{S} \oplus lab(v) \in \mathcal{A}$; □

Nos exemplos ilustrados nas Figuras 4.4(a) e 4.4(b) pode-se observar que cada nó v da árvore é representado pelo seu identificador, determinado através da função $id(v)$ que endereça cada nó da árvore de acordo com a *Ordem de Dewey* (Seção 2.3.2.3). Os rótulos dos nós são determinados pela função $lab(v)$ caso o nó seja um elemento ou um atributo, e por $val(v)$ se o nó for do tipo texto ou atributo.

4.2.2 Linguagem de Caminhos

Para descrever caminhos dentro de um documento XML, em qualquer contexto deste trabalho, será utilizada a linguagem para especificação de caminhos recomendada pelo W3C chamada XPath *(XML Path Language)* (Clark e DeRose, 1999).

Primeiramente, XPath é uma linguagem de expressão. Ela não é descrita como uma linguagem de programação ou de consulta, por isso, ela oferece funcionalidades limitadas. O uso mais comum de expressão XPath é tomar um documento XML como entrada e retornar um conjunto contendo os nós alcançáveis pela expressão.

Diz-se que um caminho p é uma sequência de rótulos $l_1/l_2/.../l_n$. Assim, uma expressão de caminho q define um conjunto $\mathcal{P}$ de caminhos. Dada uma árvore XML $\mathcal{T}$, denota-se $\mathcal{P}_\mathcal{T}$ o conjunto de caminhos em $\mathcal{T}$. Como exemplo, considere que $\mathcal{T}$ é a árvore XML ilustrada na Figura 4.4(a). O conjunto $\mathcal{P}_\mathcal{T}$ é dado por {`/`, `/nome`, `/item`, `/item/@serial`, `/item/fabricante`, `/item/modelo`, `/item/cor`, `/item/preco`}.

Pode-se observar que o operador '/' une uma sequência de rótulos (passos) dentro de um caminho. Assim, o caminho `/item/modelo` seleciona todos os elementos `modelo` que estão abaixo dos elementos `item` que descendem do nó raiz.

Em XPath uma expressão entre colchetes é chamada de predicado. Qualquer passo em uma expressão de caminho pode ser qualificado por um predicado, o qual tem a finalidade de filtrar o conjunto de nós alcançáveis. Por exemplo, a expressão `/item[fabricante = 'HP']/cor` tem como objetivo selecionar todos os elementos `cor` para todos os elementos `item` que tenham como filho um elemento chamado `fabricante` com conteúdo igual a `'HP'`.

Na linguagem de mapeamento entre fonte de dados e repositório definida em (do Nascimento e Hara, 2008) não é permitida, e nem necessária, a utilização de predicados para especificar o caminho na fonte origem e sua relação no repositório integrado. Contudo, a utilização de predicados é um recurso importante na definição do contexto em que uma regra da resolução de conflitos deve ser aplicada, assunto que será tratado nas próximas seções ao longo deste capítulo.

Para garantir a integridade da política, este trabalho utiliza apenas um subconjunto de expressões XPath para definir o contexto de uma regra de resolução. Fica definido que para expressar um caminho no repositório o administrador deve utilizar somente o operador '/' para percorrer os níveis da árvore XML. Na definição de um predicado pode-se utilizar somente nós filhos do nó que precede o predicado. Por exemplo, a expressão `/produto[fabricante = ''Sony'' and modelo = ''N450'']` estaria correta somente se os elementos `fabricante` e `modelo` fossem nós filhos de `produto` na árvore XML que os contêm. Para fins de comparação entre valores, os seguintes operadores são permitidos: '=', '>', '<', '>=', '<='. Predicados compostos por mais de uma comparação sempre devem ser descritos utilizando o operador de conjunção (*and*). Desta forma, para o restante deste trabalho, o fragmento de expressões XPath considerado será dado por $XP_{/,[\ ],and}$.

4.2.3 Repositório Integrado

No modelo definido em (do Nascimento e Hara, 2008) um repositório $\mathcal{D}$ era definido por uma árvore XML $\mathcal{T}$ e um conjunto de chaves $\mathcal{K}$. No entanto, a definição de repositório para o modelo proposto nessa dissertação precisou ser adequada devido a implementação

de uma estrutura de histórico e a utilização da política de fusão no estágio de limpeza. Ambas serão descritas ao longo deste capítulo. Assim, para esta dissertação a definição do repositório integrado de dados é dada como segue.

Definição 4.2. Um repositório integrado de dados $\mathcal{D}$, com dados importados de um conjunto $\mathcal{S}$ de árvores XML, é definido pela quíntupla $\langle \mathcal{R}, \mathcal{T}, \mathcal{K}, \mathcal{P}, \mathcal{H} \rangle$ onde

- $\mathcal{R}$ é um conjunto de pares $(s, rank)$ tal que $s \in \mathcal{S}$ representa uma árvore XML que já integra $\mathcal{D}$, e $rank$ é seu indicador de confiabilidade. O valor de $rank$ é maior para fontes com grau de confiabilidade mais elevado.

- $\mathcal{T}$ é a árvore XML que possui um conjunto $\mathcal{V}$ de nós, tal que cada nó folha $v \in \mathcal{V}$ é anotado com um conjunto de pares (id_n, p), onde id_n é o identificador de um nó n em alguma árvore $s \in \mathcal{S}$, e p é o caminho da raiz r até o nó n;

- $\mathcal{K}$ é o conjunto de chaves XML definidas para $\mathcal{T}_\mathcal{D}$. Um nó de $\mathcal{T}_\mathcal{D}$ pode ser unicamente identificado de acordo com $\mathcal{K}$ ou ter um único nó filho `S`;

- $\mathcal{P}$ indica um conjunto de decisões que aplicadas sobre $\mathcal{T}_D$ determinam como resolver conflitos ocorridos em algum $v \in \mathcal{V}$ de $\mathcal{T}_\mathcal{D}$.

- $\mathcal{H}$ é um conjunto de operações de limpeza realizadas sobre $\mathcal{T}_\mathcal{D}$ em diferentes momentos de integrações anteriores. □

4.3 Política de Fusão

Para compor um repositório integrado somente com informações consistentes, um sistema de integração de dados precisa conter um processo de limpeza de dados eficiente. Para isso, esta dissertação sugere a utilização de uma política que permita ao administrador do repositório definir regras para a resolução de conflitos nos diversos contextos do repositório. Estas regras procuram descrever as decisões comumente tomadas pelo usuário, utilizando-se algumas das estratégias para resolução de conflitos apresentadas na Seção 2.3.4.

Para resolver os conflitos detectados durante o processo de integração, este modelo baseia-se na definição de uma política de fusão, a qual consiste de um conjunto de regras para resolução de conflitos em nível de instância. Estas regras são definidas da seguinte forma:

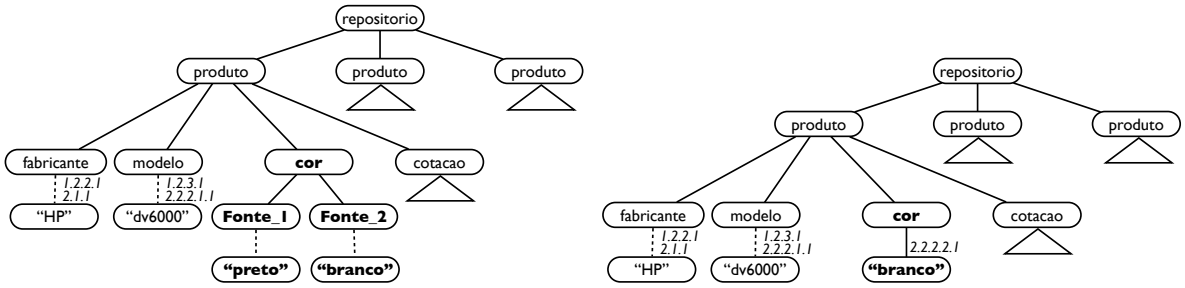
Definição 4.3. Uma *regra de resolução de conflito* é um par $\langle \sigma, \Sigma \rangle$, onde

- σ é uma expressão de caminho representando o contexto coberto pela regra;
- Σ representa uma lista não vazia de estratégias para resolução de conflitos sobre os nós alcançáveis pelo caminho determinado no contexto σ. □

O contexto de uma regra é definido pela expressão de caminho σ em $XP_{/,[\ ],and}$ e, consequentemente, ele pode cobrir não apenas um único elemento ou atributo, mas também um conjunto de nós alcançáveis através de σ. Além disso, uma regra descreve uma lista de estratégias para resolução de um conflito. Desta forma, se a primeira estratégia não é capaz de atribuir um único valor para um determinado item de dado, as estratégias seguintes são consideradas, uma após a outra, até o fim da lista ser alcançado ou o conflito ser solucionado. Se o conflito é resolvido com a aplicação de uma das estratégias, diz-se que a regra *efetivamente* resolveu o conflito de valor.

Para exemplificar a aplicação de uma regra, considere o conflito de valores ocorrido no elemento `cor` de algum `produto` do repositório, conforme ilustrado na Figura 4.5(a). Suponha que a seguinte regra foi definida na política de fusão: $\langle /produto[fabricante = \text{'HP'}]/cor, [Trust\ Your\ Friends, Cry\ With\ The\ Wolves] \rangle$. Então a estratégia *Trust Your Friends* deve ser primeiramente aplicada, seguida pela estratégia *Cry With The Wolves*. Assumindo que o grau de confiabilidade de F_2 é maior do que o de F_1, a estratégia *Trust Your Friends* é executada e o valor `'branco'`, copiado da fonte F_2, é o escolhido para ser propagado ao repositório, como ilustra a Figura 4.5(b).

No exemplo anterior, tem-se a definição de uma regra que atua sobre os conflitos que possam ocorrer em qualquer elemento `cor` de todos os produtos cujo fabricante seja `'HP'`. Com a possibilidade de definir diversas regras em diferentes contextos, o modelo deve implementar uma maneira de evitar que outras regras tenham sua ação sobre o mesmo



(a) Representação do conflito (b) Conflito resolvido

Figura 4.5: Repositório de dados antes e após a resolução do conflito no elemento `cor`.

elemento `cor`, ou seja, não deve permitir que regras distintas atuem sobre nós em comum na árvore XML do repositório.

4.3.1 Validação das Regras

A validação da política é uma tarefa realizada sempre que o usuário cria ou altera uma regra da política definida sobre o repositório integrado de dados. Esta tarefa tem o objetivo de investigar se o usuário definiu regras de resoluções conflitantes, isto é, duas ou mais regras que cobrem conjuntos diferentes de nós na árvore XML do repositório, mas que em dado momento podem agir sobre um nó em comum.

Nas definições a seguir, considere como *Nodes(r)* o conjunto de nós cobertos por uma regra de resolução r. Isto é, dada uma regra $r = (\sigma, \Sigma)$ e uma árvore XML $\mathcal{T}$, *Nodes(r)* é o conjunto de nós em $\mathcal{T}$ alcançáveis seguindo o caminho σ.

Definição 4.4. Diz-se que uma regra r_1 é uma *especialização* de outra regra r_2 se $Nodes(r_1) \subset Nodes(r_2)$. □

Definição 4.5. Dadas duas regras r_1 e r_2, diz-se que r_1 é *válida em relação* a r_2 se para qualquer estado do repositório

- $Nodes(r_1) \subset Nodes(r_2)$; ou
- $Nodes(r_1) \supset Nodes(r_2)$; ou
- $Nodes(r_1) \cap Nodes(r_2) = \emptyset$. □

Definição 4.6. Uma regra $r \in \mathcal{P}$ é dita *genérica* se seu contexto é um caminho p da forma $/l_1/l_2/.../l_n$, onde l_i é um rótulo dado por $lab(v)$ para algum $v \in \mathcal{V}$. □

Definição 4.7. Uma regra $r \in \mathcal{P}$ é dita *restritiva* se seu contexto é um caminho $p = /l_1m_1/l_2m_2/.../l_{n-1}m_{n-1}/l_n$, onde l_i é um rótulo dado por $lab(v)$ para algum $v \in \mathcal{V}$ e existe pelo menos um predicado m_i não vazio. □

Intuitivamente, as regras podem ser relacionadas por especialização ou generalização, ou ainda não relacionadas com outras regras. Seguindo a definição tradicional da hierarquia de classes, os Casos 1 e 2 permitem que as regras sejam definidas em diferentes níveis de hierarquia da árvore XML. Isto é, pode existir uma regra geral para resoluções de conflitos, mas ela pode ter sua ação sobreposta por outra regra com a finalidade de tratar nós específicos, restringindo a ação a um subconjunto de nós cobertos pela regra genérica. Estas regras podem ser chamadas de restritivas e são utilizadas no processo de limpeza com prioridade sobre as regras genéricas.

Observe na Figura 4.6 uma ilustração de como estas regras podem ser definidas sobre o repositório. Destacada pela área hachurada horizontalmente, tem-se a representação de um contexto genérico dado pela expressão XPath */produto/cor*. Esta regra indica como resolver conflitos no elemento `cor` para qualquer elemento `produto` do repositório integrado. Por sua vez, a área hachurada verticalmente sobre o elemento `cor` indica a existência de uma regra restritiva que atua sobre aquele elemento, com o contexto sendo representado pela expressão XPath */produto[fabricante='HP' and modelo='dv6000']/cor*. Perceba que no caso da regra restritiva há a utilização de um predicado na composição da expressão XPath, restringindo a resolução somente ao `produto` cujo `fabricante` seja `'HP'` e seu `modelo` seja `'dv6000'`.

No momento que ocorre um conflito em algum elemento ou atributo do repositório, as regras de resoluções são aplicadas de baixo para cima. Isto é, caso o elemento seja coberto por mais de uma regra, a resolução será determinada primeiramente pela regra mais específica. Com isso, fica garantido que um tratamento restrito sobre alguma parte do repositório será cumprido adequadamente.

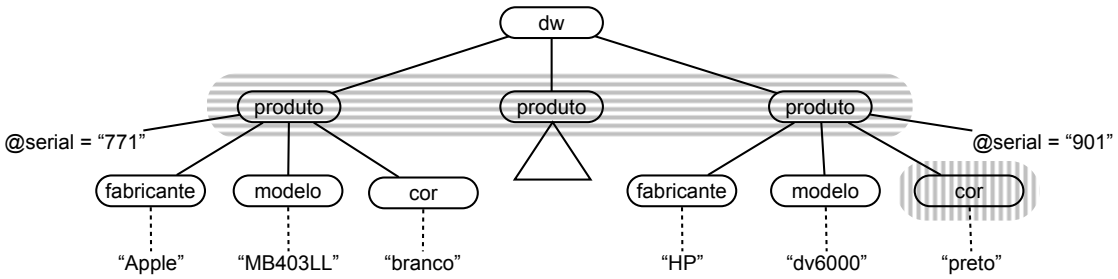


Figura 4.6: Utilização de regras genérica e restritiva sobre o elemento `cor`.

Regras que tem intersecção de cobertura e não compartilham a relação de especialização/generalização não são aceitas na política. Isso deve-se ao fato de não haver maneira determinística de saber qual regra deveria ser aplicada no momento da resolução de conflitos em nós cobertos por ambas as regras.

Para garantir a integridade da política de fusão, cada regra de resolução de conflito é validada com relação as demais regras já definidas. Essa validação envolve a verificação das relações entre os subconjuntos de nós cobertos pelo contexto de cada regra. A definição do contexto envolve a utilização de expressões XPath. Verificar a relação entre fragmentos XPath é um problema que vem sendo largamente estudado nos últimos anos (Deutsch e Tannen, 2001; Miklau e Suciu, 2004; Popa, 2002; Wood, 2003). Estes trabalhos apresentam provas formais de que alguns fragmentos de XPath podem ter sua relação verificada em PTIME, inclusive para o fragmento XPath $XP_{/,[\ ],and}$ considerado nesta dissertação.

A verificação do contexto da regras envolve ainda a presença de valores de dado na composição da expressão XPath, o que aumenta a complexidade em se determinar a equivalência entre duas expressões. Para tanto, são utilizadas informações relacionadas ao esquema do repositório para expressar a cardinalidade atribuída a cada conjunto de nós cobertos por uma regra. Isto é, com o auxílio das regras de mapeamento e o conjunto de chaves do repositório é possível inferir a cardinalidade de uma expressão XPath. As expressões que não envolvem elementos chaves tem sua resolução de equivalência não determinística, inviabilizando a criação da regra. Os algoritmos utilizados e detalhes das provas, bem como uma classificação completa sobre a complexidade em se determinar a equivalência entre expressões XPath pode ser encontrada em (Neven e Schwentick, 2006).

A definição a seguir estabelece o método utilizado para aplicação das regras de resoluções pertencentes à política de fusão.

Definição 4.8. Seja v um elemento ou atributo com valores conflitantes e $\mathcal{P}$ uma política de fusão, composta por um conjunto de regras de resolução de conflitos válidas. A regra $r_v \in \mathcal{P}$ é aplicada para solucionar um conflito de valor em v se:

- $v \in Nodes(r_v)$ e r_v efetivamente resolve o conflito de valor;
- Não existe $r_i \in \mathcal{P}$ que satisfaça a condição 1, tal que $Nodes(r_i) \subset Nodes(r_v)$; □

Considere o repositório de dados ilustrado na Figura 4.7 e as seguintes regras de resoluções:

$r_a = (/produto[fabricante = \text{'HP'} \; and \; modelo = \text{'tx1220'}]/cor, \Sigma_a)$

$r_b = (/produto[fabricante = \text{'HP'}]/cor, \Sigma_b)$

$r_c = (/produto/cotacao[loja = \text{'Fast'}]/preco, \Sigma_c)$

$r_d = (/produto[fabricante = \text{'Sony'}]/cotacao/preco, \Sigma_d)$

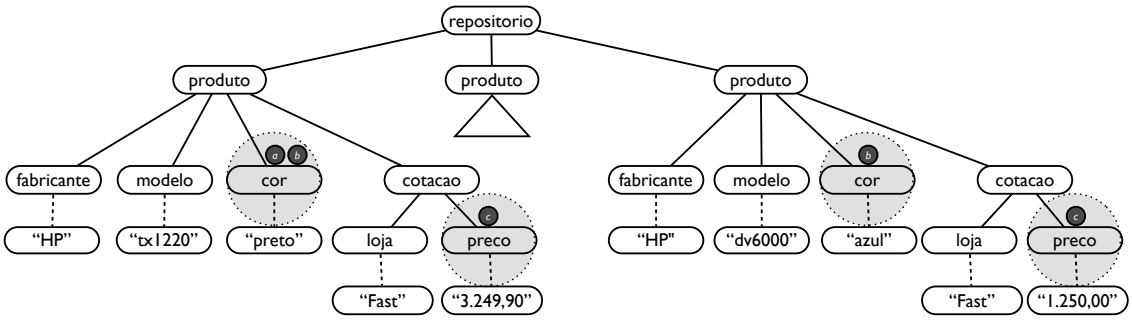


Figura 4.7: Demonstração do contexto de cobertura para as regras r_a, r_b, r_c e r_d.

As regras r_a e r_b são definidas sobre um conjunto de elementos `cor`, enquanto que r_c e r_d são definidas sobre um conjunto de elementos `preco`. Observe que r_a é uma especialização de r_b, uma vez que $Nodes(r_a) \subset Nodes(r_b)$, portanto elas são regras válidas entre si. Por outro lado, r_c não é válida com relação a r_d, pois não é possível afirmar que para todas as possíveis subárvores XML $Nodes(r_c) \cap Nodes(r_d) = \emptyset$, embora na árvore ilustrada na Figura 4.7 esta condição seja válida.

A Figura 4.8 utiliza um diagrama de Venn para facilitar a visualização da existência ou não de conflitos entre regras de resoluções. Sobre o repositório completo, ou conjunto

de todos os nós, são definidas as áreas de cobertura de cada regra de resolução. Sabendo da unicidade do subconjunto de nós cobertos pela regra r_a e conhecendo o subconjunto de nós cobertos por r_b, sua representação mostra que uma regra é válida em relação a outra. Por outro lado, o mesmo não pode ser dito com relação as regras r_c e r_d, pois o contexto de cobertura dessas regras podem, em algum momento, compartilhar nós de seus subconjuntos, isto é, o subconjunto de nós cobertos por r_c está parcialmente contido no subconjunto de nós cobertos r_d, ou vice-versa.

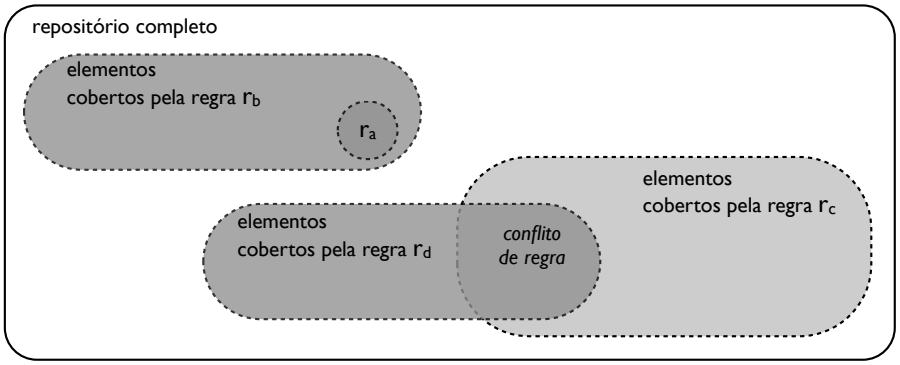


Figura 4.8: Ilustração da existência de conflito entre duas regras restritivas.

4.4 Histórico de Resoluções

Nas seções anteriores foi comentada a utilização do histórico como forma de manter informações sobre decisões tomadas por alguma regra sobre um conflito do repositório. Nesta seção, serão detalhadas as características e funcionalidades que o registro de histórico oferece ao modelo de limpeza de dados proposto neste trabalho.

O registro de histórico funciona como um repositório de dados considerados inconsistentes em determinado momento do repositório. Isto é, são dados que estiveram envolvidos em algum conflito e que por decisão da política de fusão não foram propagados ao repositório. No entanto, eles não devem ser descartados completamente, pois em uma integração futura estes dados podem ser reconsiderados e participar do processo de limpeza. Além disso, os valores descartados permanecem armazenados para permitir a reconstrução da fonte original, mantendo tanto sua estrutura como seus dados.

Definição 4.9. O *histórico de resoluções* é um conjunto de registros, onde cada registro refere-se a um dado d que foi descartado durante o processo de limpeza. Dado que d é proveniente de um elemento e de uma fonte de dados S, o registro de histórico referente a d contém os seguintes atributos:

- a *chave* que identifica o elemento ou atributo e no repositório;
- o *valor descartado* d;
- o *identificador* $id(e)$ na fonte de dados original S;
- o *caminho* da raiz até o elemento e na fonte original S;
- a *estratégia* $s \in \Sigma$ que foi aplicada para solucionar o conflito; □

Faz-se necessário manter a chave para cada elemento ou atributo que teve seu valor descartado para permitir a recuperação de todos os valores descartados para o mesmo item de dado do repositório. Estes dados podem ser úteis para realizar a reaplicação da regra de resolução em processos de limpeza futuros. Tanto a identificação do elemento quanto seu caminho na fonte da qual d foi copiado são armazenados no histórico de resoluções com a finalidade de manter a proveniência da informação. Com o armazenamento da estratégia executada para solucionar o conflito torna-se possível conhecer os motivos pelos quais o valor d foi descartado.

4.4.1 Utilização do Histórico para Reavaliações

Uma das funcionalidades proporcionadas pela manutenção de um histórico de resoluções é a possibilidade de considerar novamente um dado descartado no passado. Assim, um dado que por decisão da política de fusão foi desconsiderado, poderá em um segundo momento ter sua consistência reavaliada em uma nova integração.

Considere novamente o conflito ilustrado na Figura 4.5(a) e a regra de resolução $r = \langle /produto[fabricante = \text{'HP'}]/cor, [Trust\ Your\ Friends, Cry\ With\ The\ Wolves]\rangle$. Após a resolução do conflito, o registro para o valor `'preto'`, descartado de F_1, é armazenado no histórico de resoluções com as seguintes informações:

(*chave:* `/produto[fabricante='HP' and modelo='dv6000']/cor`, *valor:* `'preto'`, *id:* `1.2.4.1`, *caminho:* `/item/cor`, *estratégia:* `Trust Your Friends`). Perceba que a partir do valor do identificador também é possível fazer a identificação da fonte de origem, uma vez que o primeiro número da sequência corresponde ao elemento raiz, o qual coincide com o identificador da fonte.

Para elucidar a ideia de reaproveitamento dos dados no histórico, suponha que em uma nova integração a fonte F_3 é integrada ao repositório. Sabendo que esta fonte tem o mesmo grau de confiabilidade de F_2 e que F_3 informa o valor `'preto'` para o elemento `cor`, a estratégia *Trust Your Friends* não é capaz de resolver o conflito. Então, a próxima estratégia, *Cry With The Wolves*, precisa ser aplicada. Neste caso, o valor `'preto'` é escolhido para ser armazenado no repositório, uma vez que a estratégia decide pelo dado reportado pela maioria das fontes. Esta decisão só torna-se possível devido ao histórico de resoluções, pois ele mantém o valor `'preto'` do elemento `cor` descartado na integração anterior. Vale ressaltar que a utilização do histórico não se aplica a todas as estratégias, mas somente àquelas nas quais o conjunto de valores conflitantes é fundamental para o critério de decisão.

Após a última resolução, as informações referentes ao valor `'branco'`, proveniente do elemento `cor` na fonte F_2, é registrado no histórico de resoluções. Consequentemente, as informações sobre o valor reconsiderado `'preto'` deixam de existir no histórico e são atualizadas no repositório, onde terá como anotação de proveniência os identificadores nas fontes F_1 e F_3, conforme ilustra a Figura 4.9.

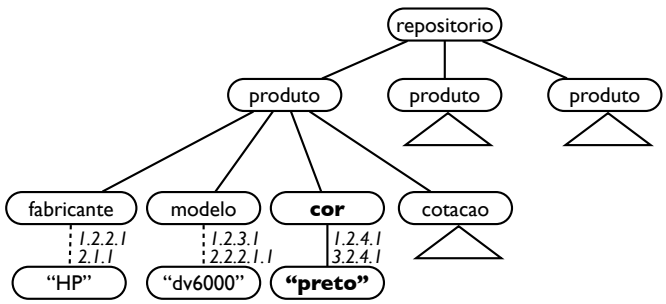


Figura 4.9: Elemento `cor` após resolução utilizando de informações do histórico.

4.5 Reconstrução das Fontes

O processo de reconstrução das fontes de dados tem como objetivo construir um documento XML que apresente sua estrutura exatamente igual àquela que originalmente foi integrada ao repositório. Neste modelo, esta funcionalidade é adequada para permitir dois tipos de reconstrução: a reconstrução com dados originais e a reconstrução com dados do repositório. A primeira é útil na medida que não exige a manutenção física do documento original e permite executar comparações com novas versões da mesma fonte que venham a integrar o repositório. A segunda reconstrução é considerada uma ferramenta de sugestão, pois a ideia é sugerir ajustes na fonte original com o intuito de evitar a reincidência de conflitos já solucionados.

4.5.1 Documento Original

A reconstrução do documento original é uma funcionalidade herdada do modelo descrito em (do Nascimento e Hara, 2008). Contudo, este processo sofreu alterações devido ao processo de limpeza de dados que este modelo implementa. Neste modelo, os dados considerados inválidos não sobrecarregam o repositório integrado. Ao invés disso, estes dados são mantidos no histórico de resoluções, apresentado na Seção 4.4.

A principal motivação para manter um processo de reconstrução da fonte original é permitir ao modelo comparar a versão integrada com uma nova versão da mesma fonte de dados. Detectadas as atualizações ocorridas, não seria necessário integrar a fonte completa novamente, pois somente os dados que sofreram modificações passariam pelo processo de integração.

No processo de reconstrução, a identificação dos dados pertencentes a fonte que está sendo construída é feita pelas anotações de proveniência existentes nos elementos folha do repositório e, para dados arquivados, pelos atributos *identificador*, *caminho* e *valor* na fonte de dados original.

A Figura 4.10 ilustra o processo de reconstrução da fonte original e ajuda a esclarecer o principal intuito dessa funcionalidade: a detecção de diferenças com a finalidade de

realizar uma integração incremental. Perceba que as partes já discutidas do sistema foram omitidas, sendo apresentadas na figura somente as partes necessárias para executar o processo de reconstrução da fonte.

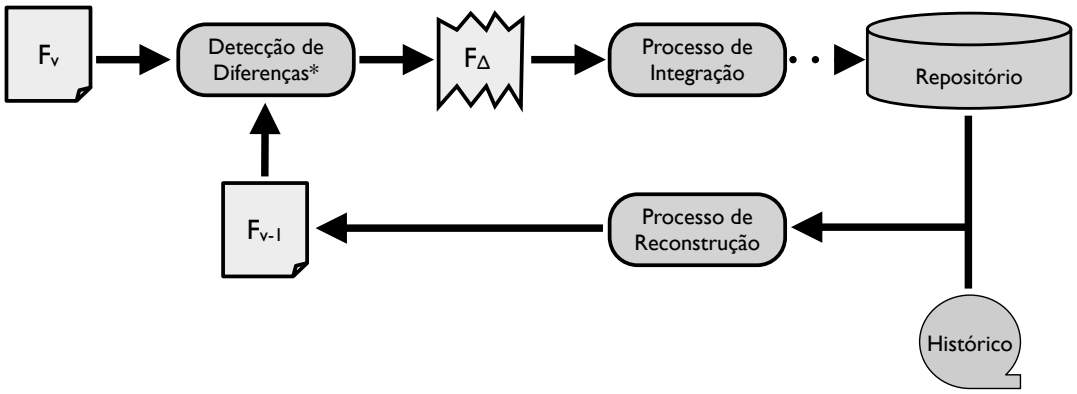


Figura 4.10: Aplicação do processo de reconstrução da fonte original.

Ao surgir uma nova versão da fonte F_v para integração, o sistema executa o processo de reconstrução da versão F_{v-1} utilizando os dados do repositório e do histórico de resoluções. Com esses dois documentos é possível executar um processo de detecção de diferenças e integrar somente as mudanças ocorridas nessa fonte de dados F_Δ. Com isso, reduz-se o tempo de integração e, consequentemente, os custos da etapa de limpeza dos dados.

4.5.2 Documento para Sugestão

Outra maneira de reduzir o tempo e o esforço na realização do processo de limpeza é, obviamente, reduzir o número de conflitos produzidos na integração.

Muitos dos conflitos que passam pelo estágio de limpeza são reincidentes e, provavelmente, já tiveram uma decisão tomada sobre eles. No entanto, como a fonte responsável pelo dado não conhece tal decisão, a fonte continua reportando o valor do dado de forma divergente. Para reduzir este ruído de integração, este modelo propõe uma reconstrução de sugestão. Isto é, reconstruir o documento com a estrutura da fonte original, mas somente com os dados considerados corretos para o domínio do repositório. Com esta possibilidade de retorno, a fonte de origem pode ajustar possíveis equívocos nos dados que a mesma informa e evitar a ocorrência de conflitos já resolvidos.

O processo de reconstrução de sugestão é similar ao processo de reconstrução da fonte original. Ambos os processos utilizam anotações do repositório e registros do histórico para formar a estrutura e os dados do documento original. No entanto, para o processo de sugestão, os valores descartados não devem recompor o documento original, mas sim os dados considerados corretos. Para isso, a reconstrução utiliza as informações do histórico, exceto o atributo *valor*, o qual deve ser consultado no repositório através do atributo *chave*.

4.6 Considerações

Neste capítulo foram apresentadas todas as funcionalidades propostas para o modelo de integração sobre um repositório de dados XML.

Inicialmente, foi apresentado o cenário de integração sobre o qual discutiu-se as funcionalidades propostas. Em seguida, foram feitas algumas definições iniciais sobre a representação dos dados utilizada no modelo, linguagem para representação de caminhos e o conceito de árvore XML.

A política de fusão foi apresentada na Seção 4.3 como uma abordagem para realizar a limpeza de dados XML durante o processo de integração. Além disso, foram discutidas as restrições aplicadas na definição das regras de resoluções, como estas devem ser descritas e de que forma são validadas.

Na Seção 4.4, é discutida a utilidade do histórico de resoluções. Uma estrutura criada para permitir a limpeza do repositório sem perder a proveniência dos dados integrados. Além disso, permite que os dados descartados sejam reaproveitados pelas estratégias em integrações futuras.

Por fim, nas Seções 4.5.1 e 4.5.2, foram discutidos os processos de reconstrução da fonte original e da fonte para sugestão, respectivamente. São processos com utilidades distintas dentro do modelo. O primeiro procura oferecer um recurso que possibilite a integração incremental a partir da detecção de diferenças entre a fonte reconstruída e uma nova versão desta mesma fonte. Já a fonte para sugestão busca retroalimentar o

sistema, pois tem o intuito de informar as fontes sobre as possíveis inconsistências em seus dados.

No próximo capítulo será apresentada a ferramenta XFusion, a qual foi utilizada para fazer a validação do modelo proposto nesta dissertação. Além disso, serão discutidos os algoritmos que foram implementados para executar as funcionalidades apresentadas neste capítulo.

Capítulo

5

XFusion

Neste capítulo é apresentada a ferramenta XFusion, um protótipo desenvolvido para validar o modelo de integração proposto. Adicionalmente, são apresentados os algoritmos responsáveis pela principais funcionalidades do sistema de integração.

5.1 Visão Geral

XFusion é uma ferramenta com interface gráfica que permite ao usuário exercitar muitas das funcionalidades desejadas em um sistema de integração padrão. A ferramenta XFusion oferece meios para visualizar os dados do repositório integrado no formato de árvore, integrar novas fontes, ilustrar conflitos ocorridos na integração, solucioná-los ou compor regras de resolução e também realizar a reconstrução das fontes seja a original ou para sugestão.

Como o foco principal desta dissertação está sobre o estágio de limpeza dos dados, a principal funcionalidade implementada na ferramenta é a composição de regras para resolução dos conflitos representados no repositório. Para isso, a ferramenta disponibiliza uma interface na qual o usuário escolhe as estratégias que irão compor a regra de resolução para determinado contexto do repositório. Através da composição destas regras o usuário evolui sua política de fusão e aumenta a área de cobertura sobre o repositório, reduzindo a ocorrência de conflitos em novas integrações.

5.2 Tecnologias Utilizadas

Para a implementação da ferramenta foram escolhidas algumas tecnologias de código aberto. A ferramenta foi desenvolvida utilizando a linguagem de programação Java[1], mais especificamente *Java™ SE Runtime Environment* na sua versão 6. Embora desenvolvido na versão 6, a ferramenta oferece compatibilidade com versões anteriores. Para desenhar e implementar a interface usou-se a API *Swing*[2] também da plataforma Java. Como ambiente integrado de desenvolvimento (IDE) foi utilizada a ferramenta livre NetBeans versão 6.8[3]. Para acessar, manipular e formatar os documentos XML utilizou-se uma solução de código aberto baseada em Java chamada JDOM[4]. Como sistema para gerenciamento e armazenamento dos dados foi utilizado eXist-db[5], uma plataforma que permite armazenar os dados de acordo com o modelo de dados XML e realizar o processamento de consultas XQuery.

A utilização destas tecnologias permitiu construir uma interface que oferece todos os recursos necessários para exercitar as funcionalidades de um sistema de integração de dados no formato XML.

5.3 Interface

Nesta seção é apresentada a interface gráfica da ferramenta XFusion e como o usuário interage a fim de realizar as atividades pertinentes ao ambiente de integração de dados. A Figura 5.1 mostra a tela principal da ferramenta, a qual é dividida em três partes: barra de ferramentas, painel do repositório e painel de fontes integradas.

Os ícones da **barra de ferramentas** representam, da esquerda para a direita, as seguintes ações:

 Abrir um repositório de dados existente;

 Criar um novo repositório de dados;

[1] http://java.sun.com/
[2] http://java.sun.com/swing
[3] http://www.netbeans.org/
[4] http://www.jdom.org/
[5] http://exist.sourceforge.net

 Integrar uma nova fonte de dados;

 Reconstruir uma fonte de dados já integrada;

 Efetuar a resolução do conflito sobre o nó selecionado;

 Navegar entre os conflitos detectados; e

 Sair do sistema.

O **painel do repositório** apresenta, em formato de árvore, as informações do repositório XML integrado, realçando os nós que apresentam conflitos em seus itens de dados. Estes nós são explicitados através da representação dos valores conflitantes juntamente com barras coloridas indicando a(s) proveniência(s) de cada valor de dado. Já o **painel de fontes integradas** informa quais fontes integram o repositório de dados. As fontes são listadas pelo seu identificador, seguido do nome da fonte e, entre parênteses, o índice de confiança atribuído a ela.

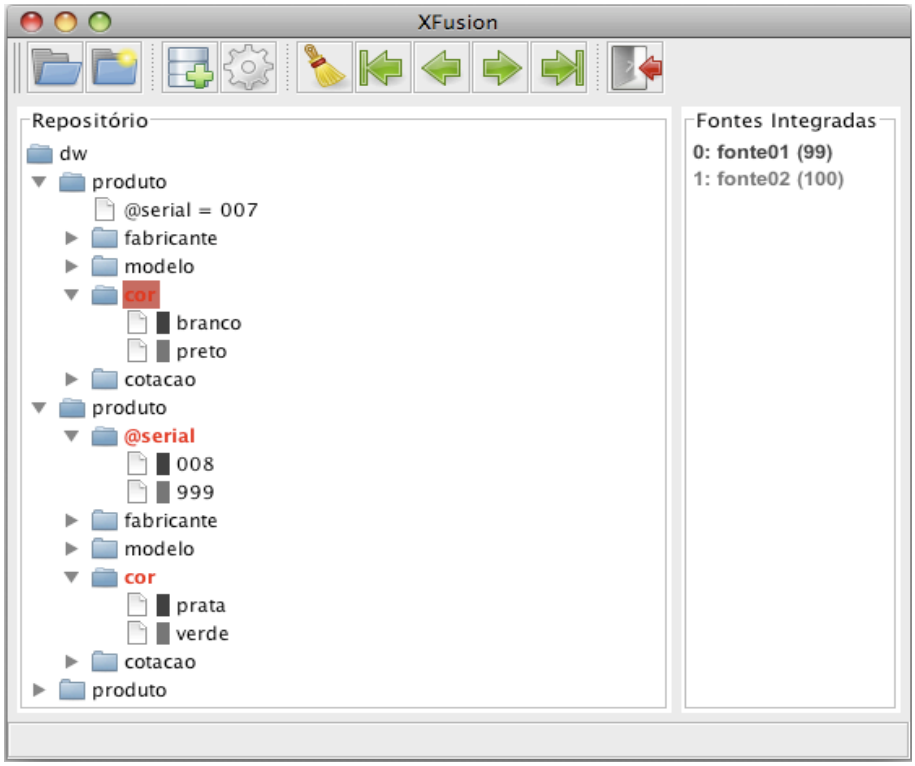


Figura 5.1: Interface principal da ferramenta XFusion.

A limpeza dos dados é a funcionalidade na qual a proposta desta dissertação mantém o foco. Portanto, a tela para resolução de conflitos, apresentada na Figura 5.2, oferece os recursos necessários para que o usuário do sistema elimine o conflito do repositório e/ou

crie e adicione na política uma regra que possa solucioná-lo. Esta tela é apresentada ao usuário sempre que ele seleciona um conflito para resolver. Neste momento, o usuário tem três opções de solução: 1) Resolver parcialmente o conflito optando por um dentre os valores conflitantes; 2) Definir um novo valor, o qual terá prioridade maior em relação aos valores informados pelas fontes; ou 3) Compor uma nova regra a partir da escolha de uma ou mais estratégias de resolução.

A ferramenta não sugere nenhuma ordem de execução das estratégias, esta ordem deve ser definida pelo usuário de acordo com seus critérios de decisão. A disposição das estratégias na ordem de execução pode acarretar decisões diferentes sobre o mesmo conflito. Por esse motivo, o usuário deve ser um especialista no domínio e monitorar o resultado da aplicação da política para efetuar eventuais ajustes e calibrações nas regras de resoluções.

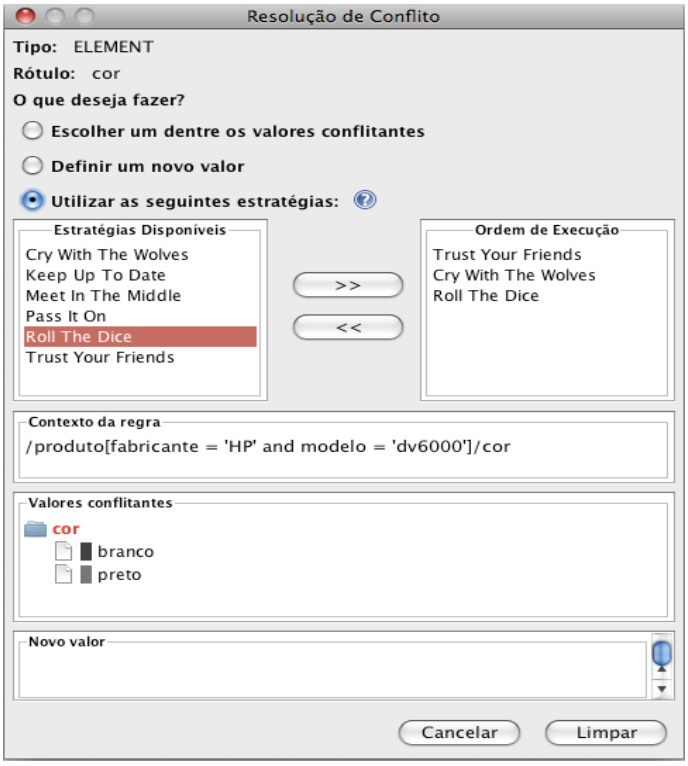


Figura 5.2: Interface para resolução de conflitos.

Caso o usuário defina uma nova regra, faz-se necessário especificar o contexto de cobertura no campo **contexto da regra**. Feito isso, o usuário clica no botão **Limpar** que realiza as tarefas de validação da regra seguida da resolução do conflito.

5.4 Algoritmos

Nesta seção são apresentados os algoritmos implementados para executar as funcionalidades do modelo proposto. Inicialmente são mostrados dois exemplos de algoritmos para estratégias e, em seguida, o algoritmo de aplicação da política de fusão. Por fim, é discutido o algoritmo de reconstrução da fonte original ou para sugestão.

5.4.1 Estratégias

Para este trabalho foram desenvolvidos algoritmos para cinco estratégias diferentes: *Cry With The Wolves*, *Keep Up To Date*, *Meet In The Middle*, *Roll The Dice* e *Trust Your Friends*. No entanto, a ferramenta permite que sejam desenvolvidas e acopladas novas estratégias de maneira simples. Para isso, é necessário desenvolver um novo algoritmo para a estratégia e incluí-lo no Algoritmo 3, a partir da Linha 13.

Definição 5.1. Dada uma entidade conflitante e, o conjunto de todos os itens de dados envolvidos no conflito em e é representado por *Valores(e)*. □

Definição 5.2. Seja v um valor pertencente a *Valores(e)*, o conjunto de fontes de dados que reportam o valor v é dado por *Fontes(v)*. □

Como exemplo das estratégias implementadas, considere os Algoritmos 1 e 2. Estes algoritmos tem a finalidade de ilustrar de maneira simples como novas estratégias podem ser elaboradas a fim de agregar novas formas de resolução de conflito ao sistema de integração. Nestes algoritmos cada entidade conflitante é representada por um objeto *Conflito*, composto pelos seguintes atributos: 1) *chave*, a qual representa a expressão XPath que identifica a entidade no repositório; 2) *valores*, que representa o conjunto de instâncias conflitantes; e 3) *escolha*, que é o conjunto de valores escolhidos após a execução de alguma estratégia.

O Algoritmo 1 implementa a estratégia *Trust Your Friends*, a qual procura escolher valores das fontes com maior grau de confiabilidade (atribuído pelo usuário no momento

da integração). Perceba que outras estratégias podem, de maneira similar, ser implementadas utilizando critérios como tamanho da fonte, quantidade de integrações recentes, proveniência, entre outros.

A lógica do Algoritmo 1 tem como base um indicador de quantidade. O indicador *max* se inicia com zero e é utilizado para guardar o maior índice de confiança entre as fontes. Desta forma, o algoritmo percorre todos os valores conflitantes (Linha 2) e para cada valor v busca nas fontes que o informaram *Fontes(v)* (Linha 3), aquela com maior grau de confiabilidade – determinado pelo atributo $S.rank$. Se uma das fontes envolvidas tem, isoladamente, o maior *rank*, então opta-se pelo valor reportado por esta fonte (Linhas 4 a 6). Caso contrário, havendo igualdade entre o grau de confiabilidade de duas ou mais fontes que informam valores distintos, o conjunto *Conflito.escolha* é composto pelos valores de todas as fontes empatadas (Linhas 7 a 9).

Algoritmo 1. *TYF(Conflito)*

Entrada: *Conflito* – Representação do conflito em uma entidade
Saída: *Conflito.escolha* – Conjunto de valores escolhidos

```
1  max ← 0
2  para cada v ∈ Conflito.valores faça
3  |  para cada S ∈ Fontes(v) faça
4  |  |  se S.rank > max então
5  |  |  |  Conflito.escolha ← {v}
6  |  |  |  max ← S.rank
7  |  |  senão se S.rank = max então
8  |  |  |  Conflito.escolha ← Conflito.escolha ∪ {v}
9  |  |  fim
10 |  fim
11 fim
```

O Algoritmo 2 recebe como parâmetro de entrada o objeto *Conflito* contendo as instâncias conflitantes e preenche o conjunto de valores escolhidos *Conflito.escolha*. Partindo do princípio que os dados corretos prevalecem em relação aos incorretos, a estratégia *Cry With The Wolves* procura encontrar o valor que é informado pela maioria da fontes, optando por esse valor em relação aos demais. Para isso, é utilizada uma variável auxilar *max*, com valor inicial zero, para controlar o valor que ocorre com maior frequência entre os valores conflitantes. Especificamente para esta estratégia, o conjunto de valores confli-

tantes *Conflito.valores* é composto tanto pelos valores da integração presente como pelos valores informados no passado e que foram desconsiderados (Linha 2).

Para cada valor conflitante é testado se o número de fontes que o informaram v é maior que max (Linhas 4). Caso afirmativo, v é o escolhido para o conjunto de escolhas (Linha 5). Já no caso em que dois ou mais valores são reportados pela mesma quantidade de fontes, todos os valores são adicionados no conjunto *Conflito.escolha* (Linhas 7 e 8). Neste caso, o conflito é parcialmente resolvido e passado, caso exista, para a próxima estratégia da regra continuar a resolução.

Algoritmo 2. *CWW(Conflito)*

Entrada: *Conflito* – Representação do conflito em uma entidade
Saída: *Conflito.escolha* – Conjunto de valores escolhidos preenchido

```
1  max ← 0
2  Conflito.valores ← Conflito.valores ∪ ConsultarHistorico(Conflito.chave)
3  para cada v ∈ Conflito.valores faça
4  |  se |Fontes(v)| > max então
5  |  |  Conflito.escolha ← {v}
6  |  |  max ← |Fontes(v)|
7  |  senão se |Fontes(v)| = max então
8  |  |  Conflito.escolha ← Conflito.escolha ∪ {v}
9  |  fim
10 fim
```

Complexidade. A complexidade dos algoritmos de estratégias implementados pode variar bastante de uma abordagem para outra. No entanto, a maior complexidade dentre os algoritmos implementados nesta dissertação é proporcional ao número de fonte integradas. Através dos Algoritmos 1 e 2 é possível observar que o pior cenário é aquele no qual o conflito envolve informações distintas entre todas as fontes integradas. Neste caso, o custo para percorrer todos os valores conflitantes é dado por $O(|S|)$, sendo S o conjunto de todas as fontes que integram o repositório. Contudo, só é possível manter o tempo proporcional ao número de fontes devido a estrutura do histórico ser baseada em uma tabela *Hash*. Esta estrutura permite acessar diretamente um registro no histórico através da sua chave (Algoritmo 2, Linha 2), o que pode ser feito em tempo constante.

5.4.2 Aplicação da Política

A aplicação da política ocorre automaticamente sempre que uma nova fonte de dados é integrada ao repositório, mas também pode ocorrer após a adição de uma nova regra na política de fusão, através da interface de resolução.

O algoritmo responsável pela execução da política, Algoritmo 3, recebe como parâmetros de entrada um conjunto $\mathcal{E}$ de entidades que apresentam conflitos em suas instâncias e uma política $\mathcal{P}$ com todas as regras de resoluções a serem aplicadas. Após a aplicação da política o resultado esperado é a redução significativa do número de inconsistências no repositório, para que o usuário consiga removê-las manualmente ou evoluir a política.

Algoritmo 3. *AplicarPolitica($\mathcal{E}$, $\mathcal{P}$)*

Entrada: $\mathcal{E}$ – Conjunto de entidades que apresentam conflitos de valores em $\mathcal{T}_\mathcal{D}$
$\mathcal{P}$ – Política de fusão, com regras ordenadas por especialização
Resultado: Repositório de dados com o mínimo de inconsistências

```
1  para cada regra r ∈ P faça
2      para cada e ∈ E ⊆ Nodes(r) faça
3          Conflito.escolha ← ∅
4          Conflito.valores ← Valores(e)
5          para cada str ∈ Σ_r faça
6              selecione str faça
7                  caso 'CWW'
8                      AplicarCWW(Conflito)          /* cry with the wolves */
9                  caso 'TYF'
10                     AplicarTYF(Conflito)          /* trust your friends  */
11                 caso 'MTM'
12                     AplicarMTM(Conflito)          /* meet in the middle  */
13                 caso
14                     ... outras estratégias ...
15                 fim
16             fim
17             GravarHistorico(str, Conflito)
18             se |Conflito.escolha| = 1 então
19                 Atualizar repositório integrado de dados
20                 parar
21             fim
22         fim
23     fim
24 fim
```

O algoritmo de aplicação percorre todas as regras da política e, para cada regra $r \in \mathcal{P}$, aplica a resolução r sobre as entidades $e \in \mathcal{E}$ que são cobertas por r. Lembrando que r é definida pelo par (σ, Σ) que representa, respectivamente, o contexto de aplicação e a lista de estratégias.

Primeiramente, o conjunto *Conflito.escolha* é definido como vazio (Linha 3), pois nenhum valor foi escolhido ainda. O conjunto *Conflito.valores* recebe todas as instâncias conflitantes em e, dada por *Valores(e)* (Linha 4). Em seguida, ocorre a aplicação, uma a uma, das estratégias configuradas na lista de estratégias da regra (Linha 6 até 15). Sempre após a execução de alguma estratégia, as informações sobre o item de dado descartado são gravadas no histórico de resoluções, juntamente com a estratégia responsável pela decisão (Linha 17). Por fim, caso haja um único valor no conjunto *Conflito.escolha*, diz-se que o conflito foi completamente resolvido. Então, o repositório é atualizado com o item de dado escolhido e passa-se para a resolução da próxima entidade conflitante (Linhas 18 até 21).

Complexidade. A complexidade computacional do algoritmo de aplicação da política de fusão é proporcional a $O(|\mathcal{P}| \cdot |V(\mathcal{T}_\mathcal{D})| \cdot |R|)$, onde $|\mathcal{P}|$ representa o tamanho da política, $|V(\mathcal{T}_\mathcal{D})|$ denota a quantidade de nós do repositório $\mathcal{T}_\mathcal{D}$ e $|R|$ representa o tempo total de aplicação de uma regra da política. Observe que $\mathcal{P}$ consiste de um conjunto de regras $r = (\sigma, \Sigma)$ e assim o tamanho da política $|\mathcal{P}|$ é dado pelo somatório de σ_r e Σ_r para cada regra r. Como verificado anteriormente, dentre as estratégias implementadas a maior complexidade tem custo proporcional ao número de fontes integradas, dado por $O(|S|)$. Para cada regra r, no pior caso há a ocorrência de conflitos em todos os nós da árvore XML do repositório, o que pode ser determinado em tempo $O(V(\mathcal{T}_\mathcal{D}))$ uma vez que é percorrido cada nó $v \in \mathcal{T}_\mathcal{D}$. Assim, considerando as estratégias implementadas, pode-se dizer que o custo para aplicação da política denotado por $O(|\mathcal{P}| \cdot |V(\mathcal{T}_\mathcal{D})| \cdot |R|)$ é equivalente a $O(|\mathcal{P}| \cdot |V(\mathcal{T}_\mathcal{D})| \cdot |S|)$.

5.4.3 Reconstrução da Fonte Original

A reconstrução da fonte original no sistema de integração proposto nesta dissertação tem dois objetivos principais. Primeiro, reconstruir o documento integrado com a estrutura e os dados que foram originalmente informados, processo útil tanto para reduzir espaço de armazenamento como para permitir a integração incremental. Segundo, construir um documento para sugestão, isto é, com a estrutura original, mas com os dados do repositório integrado, que não necessariamente são provenientes da fonte a ser reconstruída. A sugestão serve para informar a fonte original os que estão sendo considerados corretos, retroalimentando todo o sistema de integração.

Para atender estes dois objetivos, foi utilizado como base o algoritmo proposto por (do Nascimento e Hara, 2008). Assim, um novo código foi implementado conforme mostrado no Algoritmo 4. Devido a nova arquitetura do repositório de dados, na qual os dados "incorretos" são mantidos em uma estrutura separada, o Algoritmo 4 precisa dos seguintes parâmetros de entrada: 1) a árvore XML $\mathcal{T}_{\mathcal{D}}$ que representa o repositório de dados; 2) o histórico de resoluções $\mathcal{H}_{\mathcal{D}}$ com os dados descartados em resoluções passadas sobre o repositório $\mathcal{D}$; 3) o identificador id_S, que identifica a fonte S a ser reconstruída; e 4) o *tipo* de reconstrução desejado, se original ou se para sugestão. Como saída, o Algoritmo 4 devolve a árvore XML $\mathcal{T}_S$ com os valores de seus nós carregados de acordo com o tipo de reconstrução.

Inicialmente cria-se o nó raiz r da árvore XML $\mathcal{T}_S$ (Linha 1). Em seguida, através da função *Folhas*($\mathcal{T}_S$), identifica-se o conjunto de nós folha da árvore XML $\mathcal{T}_S$, estejam eles na árvore do repositório de dados $\mathcal{T}_{\mathcal{D}}$ ou tenham sido descartados e estejam armazenados no histórico de resoluções $\mathcal{H}_{\mathcal{D}}$ (Linhas 2 a 4).

Quando um nó folha f está presente no repositório, ele pode ser identificado pela anotação `@prov` que é composta pelo par $(id(f), caminho(f))$ que representam, respectivamente o identificador *Dewey* de f e o caminho da raiz até f na fonte original S. Já quando um nó folha está no histórico de resoluções, ele é identificado pelo atributo `id` do registro de histórico. Desta forma, quando deseja-se conhecer os nós folha provenientes da

fonte S, aplica-se a função $prim(id(f))$ sobre o identificador de f para descobrir o primeiro número do identificador. Quando este número é igual ao id_S, diz-se que $f \in$ *Folhas*($\mathcal{T}_S$).

Algoritmo 4. *ReconstruirFonte*($id_S, tipo$)

Entrada: $\mathcal{T}_D$ – Árvore XML do repositório de dados $\mathcal{D}$
$\mathcal{H}_D$ – Histórico de resoluções do repositório $\mathcal{D}$
id_S – Identificador da fonte de dados
tipo – Indicador do tipo de reconstrução (original ou sugestão)
Saída: $\mathcal{T}_S$ – Árvore XML da fonte de dados original ou de sugestão

```
1  V(T_S) ← {r}
2  Folhas(T_D) ← {n ∈ V(T_D) | n é um nó folha e ∃ n.@prov onde prim(id(n)) = id_S }
3  Folhas(H_D) ← {n ∈ H_D | n é um registro do histórico, no qual prim(n.id) = id_S}
4  Folhas(T_S) ← Folhas(T_D) ∪ Folhas(H_D)
5  para cada f ∈ Folhas(T_S) faça
6      anc ← {n ∈ T_S | id(n) é o prefixo mais longo de id(f)}
7      Seja id(anc) o identificador do nó ancestral na forma j_0.j_1.....j_a
8      Seja id(f) o identificador do nó corrente na forma j_0.....j_a.j_{a+1}.....j_{a+m}
9      Seja caminho(f) o caminho do nó f até a raiz r de S na forma /l_1/l_2/.../l_{a+m}
10     para i = 1 até m faça
11         Crie o nó n_i em T_S como um filho de anc com o rótulo l_{a+i}
12         id(n_i) ← id(anc).....j_{a+i}
13         anc ← n_i
14     fim
15     se tipo = original então
16         val(n_i) ← val(f)
17     senão
18         val(n_i) ← consulta valor de f no repositório de dados através de f.chave
19     fim
20     se n_1 é um nó do tipo elemento então
21         Ordene a lista ele(a) de acordo com os identificadores dos nós
22     fim
23 fim
24 retornar T_S
```

Determinados os elementos folha da fonte S, inicia-se a reconstrução da estrutura do documento original. Observe que a partir $id(f)$ e $caminho(f)$ é possível reconstruir a estrutura original do nó raiz até o nó f em S. Isto é, este caminho segue os nós $n_1, ..., n_{a+m}$, tal que para cada n_i, $id(n_i) = j_0.....j_i$ e $lab(n_i) = l_i$ (Linhas 10 a 14). O último nó do caminho é o próprio nó f, sendo que provém deste o valor inserido em S originalmente. Ou, caso seja uma reconstrução para sugestão, o valor é consultado no repositório integrado (Linhas 15 a 19). Após a inclusão do nó f em $\mathcal{T}_S$, faz-se uma

ordenação, através dos identificadores *Dewey*, na lista de elementos irmãos de f para que se mantenha a ordem relativa dos elementos existente no documento original (Linhas 20 a 22).

Complexidade. O algoritmo de reconstrução tem sua complexidade determinada por $O(|V(\mathcal{T}_\mathcal{D})|^2 + |\mathcal{H}_\mathcal{D}|)$. Isto é comprovado através da obtenção dos nós folha do repositório integrado de dados provenientes de uma determinada fonte (Linha 2) adicionado do gasto proporcional ao tamanho do histórico para recuperar as informações descartadas (Linha 3), tarefas que são realizadas em tempo $O(|V(\mathcal{T}_\mathcal{D})| + |\mathcal{H}_\mathcal{D}|)$. Em seguida percorrem-se todos os nós folha, tempo dado por $O(|V(\mathcal{T}_\mathcal{D})|$. Para cada nó é reconstruído seu caminho até a raiz de $\mathcal{T}_\mathcal{D}$. O caminho de n_i até r é no pior caso a altura da árvore XML, que é $O(|V(\mathcal{T}_\mathcal{D})|)$. Logo, a complexidade da reconstrução de todos os caminhos é dada por $O(|V(\mathcal{T}_\mathcal{D})|^2)$. Desta forma, tem-se a complexidade dada por $O(|V(\mathcal{T}_\mathcal{D})| + |\mathcal{H}_\mathcal{D}|) + O(|V(\mathcal{T}_\mathcal{D})|^2) = O(|V(\mathcal{T}_\mathcal{D})|^2 + |\mathcal{H}_\mathcal{D}|)$.

5.5 Considerações

Neste capítulo foi apresentada a ferramenta XFusion, a qual permite validar todas as funcionalidades implementadas no modelo proposto nesta dissertação. Foi mostrada a sua interface e como o usuário interage para realizar a limpeza dos dados e definir novas regras de resoluções. Além disso, foram discutidos os algoritmos responsáveis pelas ações de aplicação da política e reconstrução dos documentos originais ou de sugestão.

Com a implementação da XFusion, tem-se a validação do modelo proposto e a possibilidade de exercitar as atividades de um sistema de integração. Então, é possível traçar uma análise comparativa com relação aos sistemas de integração apresentados no Capítulo 3. Um resumo desta análise é mostrado na Tabela 5.1.

Os repositórios armazenados no eXist-db seguem a abordagem *Local-as-View*, pois o esquema do repositório é único e cada fonte tem um mapeamento relacionado com o esquema do repositório. Cada repositório mantém um arquivo de chaves para XML, as quais tem o objetivo de auxiliar na detecção de duplicidades no momento da integração.

Os mapeamentos, bem como as chaves, são definidos manualmente pelo usuário em um arquivo XML à parte.

Tabela 5.1: Comparação entre XFusion e outros sistemas de integração.

Sistema	Modelo de Dados	Modelo de Integração	Estratégias de Resolução	Especificação da Fusão	Detecção de Duplicidade
Fusionplex	Relacional	GLaV	*Keep to up date*	Manualmente, via consulta	Assume ID global
HumMer	Relacional	GaV	*Keep up to date, Trust your friends, Meet in the middle, Pass it on, Roll the dice, ...*	Manualmente, via consulta	ID global gerado por algoritmo próprio
Potter`s Wheel	Relacional	n/a	*Pass it on*	Manualmente, via transformações	n/a
AJAX	Relacional	n/a	*Keep up to date, Trust your friends, Meet in the middle* e outras definidas pelo usuário	Semiautomaticamente, via fluxo de transformações	ID global gerado por funções de similaridade
TSIMMIS	OEM	GaV	*Trust your friends*	Manualmente, via regras no mediador	Assume ID global
Nimble	XML	Desconhecido	*Pass it on*	Manualmente	Tabela de mapeamento
XClean	XML	n/a	Funções definidas ou acopladas pelo usuário	Semiautomaticamente, via programação XClean	ID global gerado por funções de similaridade
XFusion	XML	LaV	*Keep up to date, Trust your friends, Meet in the middle, Pass it on, Roll the dice, ...*	Semiautomaticamente, via política de fusão	ID global, dado por chave XML

Perceba que o modelo proposto e validado via XFusion é o único sistema para dados XML que implementa estratégias para fusão de dados. A especificação da fusão não é completamente automatizada, pois o usuário é quem mantém a política de fusão. No entanto, quanto se tem uma política bastante evoluída e com cobertura total sobre o repositório, a resolução de conflitos pode ser realizada completamente sem a intervenção humana. Com relação a identificação de dados duplicados, no modelo proposto é utilizado o conceito de chaves para XML para realizar esta tarefa, enquanto que os outros dois sistemas de integração para dados XML, Nimble e XClean, utilizam tabela de mapeamentos e funções de similaridade, respectivamente.

Em suma, pode-se dizer que o modelo proposto nesta dissertação, juntamente com a ferramenta XFusion, são importantes contribuições na área de integração de dados XML. Da mesma maneira que procuram garantir a qualidade e a consistência dos dados no repositório, oferecem recursos ao sistema de integração como manutenção de proveniência e reconstrução do documento original.

Capítulo

6

Experimentos

Neste capítulo são apresentados os resultados dos experimentos realizados para validação do modelo proposto. Foram exercitadas as funcionalidades desenvolvidas e agregadas no sistema de integração sugerido nesta integração, principalmente no que diz respeito à aplicação da política de fusão. Inicialmente, é apresentado o ambiente considerado para a realização dos experimentos. Em seguida, são discutidos os resultados do processo de aplicação da política de fusão, as dimensões do repositório e, por fim, o desempenho do algoritmo de reconstrução da fonte original e de sugestão.

6.1 Configuração

Para a realização dos experimentos foi utilizado um computador pessoal, executando o sistema operacional *Mac OS X 10.5.8*, equipado com um processador *Intel Core 2 Duo* 2.4 Gigahertz e com 2 Gigabytes de memória primária.

Todas as interações e experimentos foram realizados via ferramenta XFusion e tiveram os repositórios armazenados no banco de dados eXist-db. Foram utilizadas 50 fontes de dados com informações sobre produtos eletrônicos, que somados representam mais de 10000 itens, totalizando 8250 KB de dados. Todas as fontes são de dados considerados sintéticos, pois foram criados para atender condições específicas e certas necessidades que podem não ser encontradas facilmente em dados reais. Por exemplo, para medir o

desempenho da aplicação da política foi necessário criar certos percentuais de conflitos que seriam difíceis de repetir em dados reais.

6.2 Aplicação da Política de Fusão

Nesta seção são apresentados os testes referentes ao desempenho no estágio de limpeza do modelo de integração proposto. O objetivo dos testes é verificar o tempo de aplicação da política de fusão em relação ao tamanho do repositório e o número de conflitos a resolver.

O experimento foi realizado com a aplicação da política de fusão sobre o repositório já integrado, isto é, com as fontes integradas e as inconsistências representadas no repositório. Neste contexto, três cenários foram exercitados com a finalidade de mensurar o comportamento dos algoritmos. Primeiro, a política foi aplicada em um repositório no qual 8% de seus itens de dados apresentavam conflitos. Posteriormente, a política foi aplicada em repositórios com, respectivamente, 15% e 25% de conflitos em suas entidades. O índice de 8% de conflitos foi determinado a partir da média de conflitos encontrados durante as integrações entre fragmentos da base DBLP realizadas através do modelo de integração desenvolvido por (do Nascimento e Hara, 2008). Já os índices de 15% e 25% foram escolhidos apenas com a finalidade de medir o desempenho da aplicação na medida que crescem os percentuais de conflitos no repositório.

A política de fusão foi definida de forma a abranger todos os nós do repositório e utilizou regras compostas, em sua maioria, pelas estratégias *Trust Your Friends*, *Cry With The Wolves* e *Roll The Dice*. Cada aplicação da política foi executada sobre 6 repositórios de tamanhos distintos para analisar as consequências tanto do aumento do número de conflitos quanto do número de nós folha armazenados no repositório.

Os cenários foram criados através da integração de 5 fontes de dados artificialmente montadas para resultar no percentual de conflitos desejado, uma vez que o objetivo foi testar o desempenho obtido pela aplicação da política.

O gráfico apresentado na Figura 6.1 mostra os resultados coletados após o exercício do experimento. Em um repositório composto por 10000 nós folha dos quais 8% apresentavam conflitos, percentual próximo ao encontrado em integrações reais, a política foi aplicada

em 43 segundos. Neste mesmo repositório, com percentuais de 15% e 25% de conflitos a política levou, respectivamente, 74 e 183 segundos para fazer a limpeza das inconsistências. Este desempenho é considerável comparado à resolução manual, pois permite solucionar, em média, um conflito a cada 58 milissegundos. Já a resolução manual gastaria, em média, 10 segundos por conflito solucionado, levando-se em consideração que o usuário que toma as decisões é um especialista no domínio.

No pior desempenho deste experimento, a aplicação levou 324 segundos para solucionar 2400 conflitos em um repositório com 30000 nós folha. Isto representa a resolução de um conflito a cada 135 milissegundos. No entanto, considerando todos os cenários exercitados, o tempo médio gasto na aplicação da política de fusão foi de 60 milissegundos por conflito resolvido e a pequena variação entre as aplicações justifica-se pelas execuções das estratégias definidas na política.

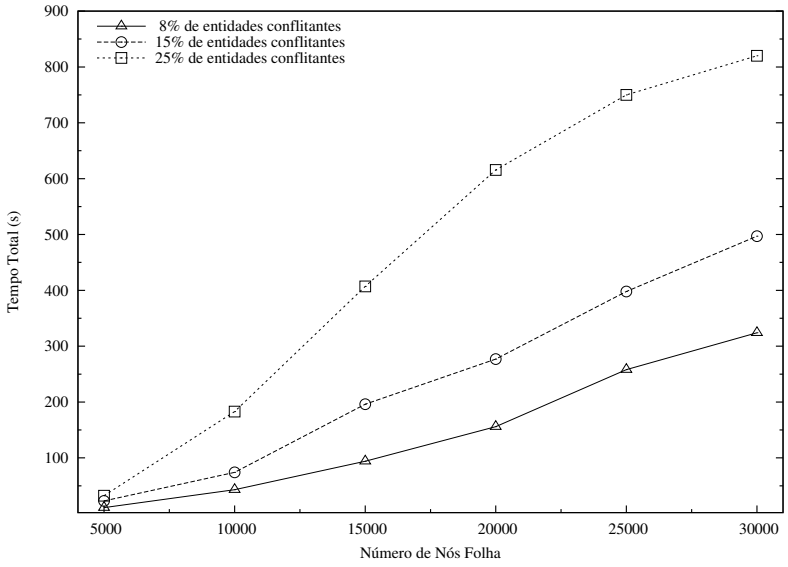


Figura 6.1: Tempo total gasto na aplicação da política de fusão.

Em outro experimento executado, pretendeu-se medir o impacto do número de fontes integradas sobre o tempo total de aplicação da política de fusão. Este teste é importante já que o aumento na quantidade de fontes integradas promove uma variação no número de valores envolvidos em um conflito. Esta variação exige mais dos algoritmos na execução das estratégias e, consequentemente, causa impactos na aplicação da política.

Para medir os efeitos da escalabilidade das fontes foram considerados 5 cenários, nos quais o repositório de dados mantinha, respectivamente, 10, 20, 30, 40 e 50 fontes armazenadas. Em todas as execuções deste teste foram considerados repositórios com 8% de entidades conflitantes.

O gráfico da Figura 6.2 apresenta os impactos identificados conforme aumenta-se número de fontes que integram o repositório. Observou-se que a quantidade de fontes de dados integradas tem influência sobre o tempo de execução da política de fusão. No entanto, o acréscimo no tempo de resolução é relativamente pequeno, pois nos testes efetuados a variação ficou entre 5% e 10% sobre o tempo de execução, conforme aumentava-se o número de fontes no repositório.

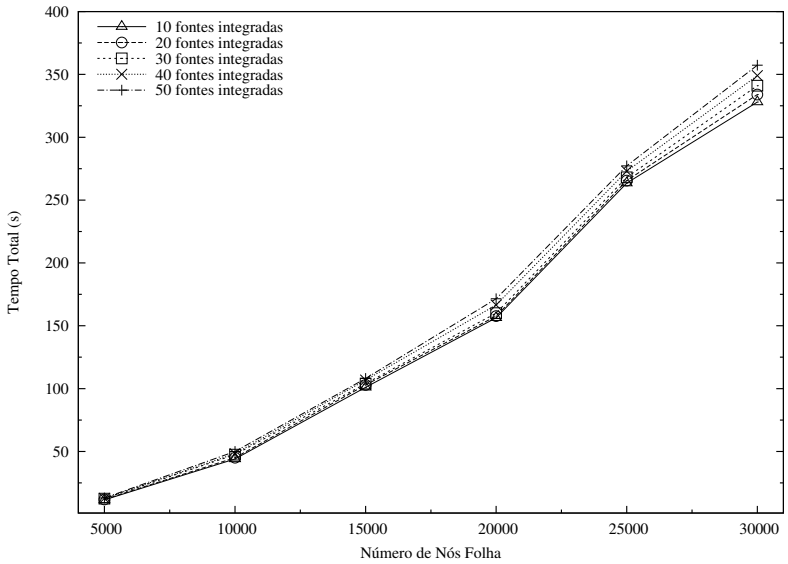


Figura 6.2: Tempo total despendido na aplicação da política de fusão.

O desempenho da aplicação da política nos diversos cenários exercitados foi satisfatório em relação ao tempo gasto na limpeza manual dos dados. Observou-se que o tempo de aplicação cresce proporcionalmente ao número de nós folha armazenados no repositório e também ao percentual de conflitos detectados. Além disso, notou-se que a escalabilidade das fontes no repositório não causa grandes perdas no tempo de aplicação da política, pois nos testes realizados este prejuízo foi de no máximo 10%.

Alguns casos testes do tipo caixa-preta foram realizados para verificar o resultado obtido pela aplicação da política. O teste aplicado consistia de quatro passos:

1. Definir os dados corretos entre as fontes;

2. Definir a política de fusão, com estratégias distintas;

3. Integrar as fontes e aplicar a política;

4. Comparar os valores propagados ao repositório com aqueles definidos no passo 1.

Após a execução destes casos de testes pode-se observar que mais de 90% dos conflitos são solucionados de acordo com o esperado. Este número justifica-se na medida que as estratégias implementadas são em sua maioria baseadas em um critério claro e exato, tornando o resultado previsível e conhecido. As exceções ficam para aqueles conflitos solucionados pelas estratégias *Roll The Dice* e *Meet In The Middle*. Como os testes foram realizados com dados sintéticos, optou-se por não apresentar maior detalhamento dos resultados com relação a qualidade das decisões tomadas.

6.3 Dimensão do Repositório

Como o modelo proposto nesta dissertação permite a reconstrução das fontes de dados integradas, não é necessário manter seus documentos originais. Assim, o objetivo deste experimento é quantificar a economia de espaço obtida após a limpeza do repositório, porém com a manutenção dos dados descartados em um histórico de resoluções.

Para realizar este experimento foram integradas 3 fontes de dados em um novo repositório. Inicialmente, a integração foi realizada pelo processo herdado de (do Nascimento e Hara, 2008), o qual mantém no repositório inconsistências temporárias e anotações de proveniência em nós do tipo folha. Com isso, realizou-se a medição do tamanho do repositório com os conflitos explícitos e os nós folha anotados.

Em seguida, aplicou-se a política de fusão para solucionar os conflitos e permitir mensurar o tamanho do repositório limpo e com os dados descartados mantidos no histórico de resoluções. Por fim, descontadas as anotações do repositório e somando-se o tamanho das fontes originais realizou-se a medição do repositório inalterado, ou seja, com conflitos detectados e com a manutenção de cópias dos documentos originais integrados.

O resultado dessas três medições pode ser observado no gráfico da Figura 6.3. Percebe-se que há um ganho de 20% a 30% de espaço quando deixa-se de manter cópias das fontes originais e opta-se por fazer anotações em nós folha. No entanto, quando ocorre a limpeza dos dados a economia é ainda maior, pois as inconsistências são eliminadas e não são mais representadas na árvore do repositório.

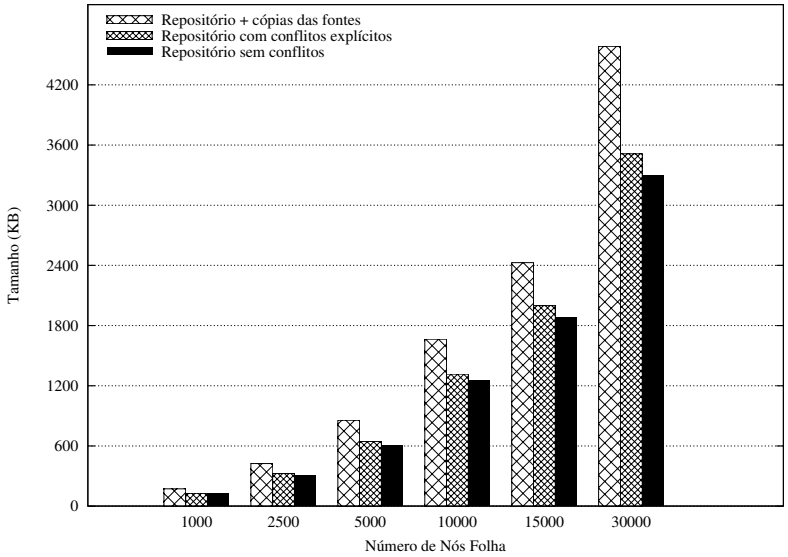


Figura 6.3: Espaço utilizado no armazenamento dos dados integrados.

O repositório limpo, com os dados descartados mantidos em uma base histórica, apresenta um leve ganho em relação ao repositório com representação dos conflitos, reduzido em média 3% o espaço de armazenamento. Por consequência, há uma economia de 23% a 33% em relação ao espaço utilizado em um repositório padrão com cópias das fontes. Perceba que o percentual de espaço economizado cresce proporcionalmente ao tamanho do repositório. Isto é, quanto maior o repositório possivelmente maior será a economia de espaço, pois haverá mais documentos ou documentos maiores que serviram de fontes de dados para o repositório integrado.

6.4 Reconstrução da Fonte

O processo de reconstrução baseia-se no repositório integrado, no histórico de resoluções e no mapeamento fonte-repositório. O experimento foi realizado com o objetivo de verificar o tempo gasto para reconstruir uma fonte de dados integrada na sua forma original ou

como documento de sugestão. Para isso, considerou-se o maior repositório de dados utilizado nos experimentos, com 30000 nós folha, e foram reconstruídas seis fontes de tamanhos distintos.

O desempenho do algoritmo de reconstrução é ilustrado no gráfico da Figura 6.4. Este gráfico apresenta a evolução do tempo de reconstrução da fonte original em comparação com o tempo para reconstruir um documento de sugestão. Perceba que os tempos de reconstrução da fonte original e de sugestão são muito semelhantes, sendo que a reconstrução para sugestão tem um pequeno prejuízo, devido ao acesso adicional que é feito ao repositório para sobrescrever o valor original da fonte.

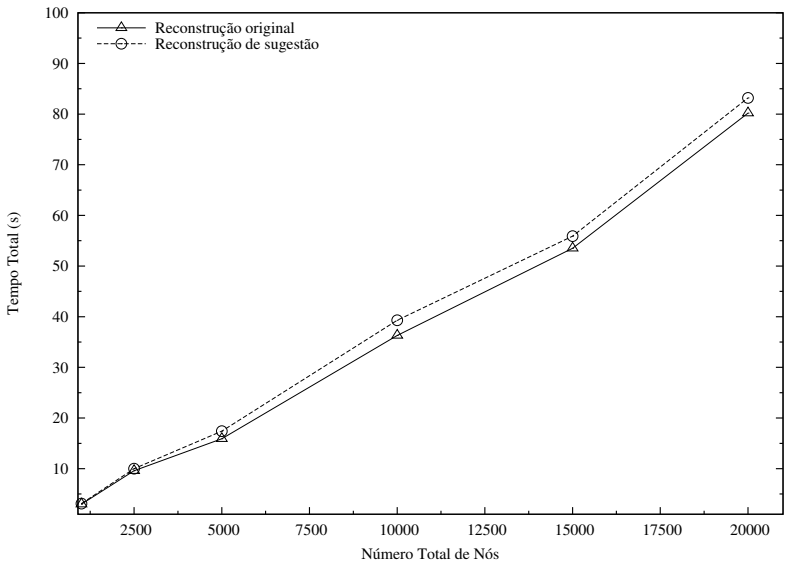


Figura 6.4: Tempo de reconstrução do documento original e para sugestão.

Como esperado, o tempo de reconstrução é maior quanto maior a quantidade de nós a serem reconstruídos. Analisando o Algoritmo 6.4 e os resultados obtidos, entende-se que o crescimento do tempo é relativo ao número de elementos que o sistema gerencia em memória. Para reduzir ainda mais este tempo existe a possibilidade de realizar a reestruturação das fontes, conforme sugerido no trabalho de (do Nascimento e Hara, 2008).

6.5 Considerações

Neste capítulo foram apresentados os resultados obtidos pelos algoritmos, ao exercitar, através do XFusion, as funcionalidades do modelo de integração de dados apresentado no Capítulo 4. Os dados foram integrados considerando o processo de integração proposto em (do Nascimento e Hara, 2008) e, a partir dessas integrações, foram testadas as funcionalidades implementadas nesta dissertação, principalmente em relação ao desempenho dos processos.

Inicialmente foi verificado o tempo de resolução dos conflitos a partir da aplicação da política de fusão. A política de fusão é considerada uma alternativa para manter o repositório de dados limpo e consistente. Para tanto, sua aplicação não deve onerar o processo de integração e limpeza dos dados. Com esse objetivo, pode-se dizer que o processo atigiu bons índices de desempenho, sofrendo pequenos impactos tanto com relação ao crescimento do repositório, quanto a escalabilidade do número de fontes integradas.

Quanto ao espaço utilizado para armazenamento dos dados integrados, o modelo manteve os índices computados no modelo de (do Nascimento e Hara, 2008) e houve uma economia de até 5% em alguns casos. A princípio este número não é significativo, mas levando-se em conta que o modelo desta dissertação adiciona uma estrutura específica para dados incorretos, a qual permite tomar decisões futuras e reconstruir fontes originais, admite-se como relevante a redução no espaço de armazenamento.

Por fim, mediu-se o desempenho dos processos de reconstrução da fonte integrada. Para reconstruções da fonte original, obteve-se um desempenho linear ao tamanho da fonte e independente ao tamanho do repositório. A reconstrução para sugestão seguiu a mesma curva de desempenho, com uma penalização aceitável devido ao tempo para correção do dado original. Infelizmente, não havia parâmetros para comparação, pois não foram encontrados na literatura sistemas que oferecessem funcionalidade semelhante.

Capítulo

7

Conclusões

Esta dissertação teve seu foco no processo de limpeza sobre um repositório integrado de dados XML. Para isso, foram consideradas as característica do modelo de integração proposto por (do Nascimento e Hara, 2008) e sobre ele foram aplicadas as funcionalidades propostas nesta dissertação.

No cenário de integração anterior, a participação do usuário na resolução de conflitos era constante. Com a implantação da política de fusão e a possibilidade de definir regras de resolução de conflito sobre sub-árvores do repositório, reduziu-se a quantidade de intervenções do usuário no processo de limpeza. Os conflitos detectados em integrações subsequentes, e que estão dentro do contexto das regras existentes, podem ser automaticamente solucionados.

O histórico de resoluções é outra parte fundamental do modelo proposto, pois esta estrutura mantém a proveniência dos dados descartados do repositório, possibilitando sua reutilização em aplicações seguintes da política de fusão. Além disso, as informações do histórico de resoluções são fundamentais para garantir as funcionalidades de reconstrução da fonte original e da fonte para sugestão.

O modelo foi validado através do desenvolvimento da ferramenta XFusion e do exercício de testes de desempenho. A XFusion foi desenvolvida para permitir realizar todas as atividades pertinentes ao sistema de integração. Assim, uma vez criado um repositório

com fontes integradas, tornou-se possível testar as funcionalidades de aplicação da política, reconstrução das fontes e, ainda, verificar o espaço de armazenamento total utilizado pelo repositório materializado.

Para viabilizar a utilização do modelo, os experimentos foram realizados com dados preparados para simular os cenários de integração desejados. O desempenho da aplicação da política de fusão foi bastante satisfatório, principalmente se for considerado o tempo que se gastaria para solucionar manualmente os conflitos. Em média, a política de fusão realizou a resolução de um conflito a cada 58 milissegundos. Outro fator analisado foi a escalabilidade de fontes integradas e seus efeitos na resolução dos conflitos. Neste cenário, percebeu-se que o número de fontes no repositório influencia linearmente o tempo de execução da política de fusão.

As tarefas de reconstrução da fonte original e para sugestão também tiveram desempenho dentro do esperado. O processo de construção do documento de sugestão manteve tempos próximos ao de reconstrução da fonte original, ocorrendo uma penalização de aproximadamente 6% devido ao acesso adicional ao repositório. Este acesso é necessário para encontrar o valor sugerido para o atributo da fonte original.

O resultado dos experimentos e a utilização da XFusion justificaram a utilização do modelo como forma de realizar a integração e limpeza de dados no formato XML em um repositório materializado.

Além da ferramenta XFusion, desenvolvida para validar o modelo, esta dissertação deu origem às seguintes publicações:

- CECCHIN, Frantchesco; HARA, Carmem S. **Resolução de Conflitos em Documentos XML**. VIII Workshop de Teses e Dissertações em Bancos de Dados, realizado juntamente com o Simpósio Brasileiro de Banco de Dados, Fortaleza, Ceará, Brasil, 2009.

- CECCHIN, Frantchesco; CIFERRI, Cristina D. A.; HARA, Carmem S. **XML Data Fusion**. In proceedings of 12th International Conference on Data Warehousing and Knowledge Discovery, Bilbao, Espanha, 2010.

7.1 Trabalhos Futuros

A partir da definição e implementação do modelo descrito nesta dissertação, entende-se que cabem como extensão ao modelo os seguintes trabalhos futuros:

- Integração deste modelo com uma técnica de atualização incremental do repositório de dados quando novas versões das fontes de dados estão disponíveis. Para isso, seria utilizado o recurso de reconstrução da fonte original comparado com a nova versão da fonte de dados.

- Extensões para a política de fusão, oferecendo novas estratégias e uma definição declarativa na aplicação das regras de resolução. Por exemplo, permitir a execução condicional das estratégias.

- Implementar um sistema de classificação das fontes de dados integradas, isto é, um algoritmo acoplado ao modelo de integração que premie fontes com dados de alta qualidade. Isso possibilita a evolução dos critérios de limpeza e automatiza a atribuição de prioridades.

- Agregar novas funcionalidades a ferramenta XFusion, oferecendo recursos como uma interface para execução de consultas ao repositório, edição da política e definição de mapeamentos.

Referências Bibliográficas

Berlin, J.; Motro, A. Autoplex: Automated discovery of content for virtual databases. In: *9th International Conference on Cooperative Information Systems*, 2001, p. 108–122.

Bilke, A.; Bleiholder, J.; Naumann, F.; Böhm, C.; Draba, K.; Weis, M. Automatic data fusion with HumMer. In: *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, VLDB Endowment, 2005, p. 1251–1254.

Bishr, Y. Overcoming the semantic and other barriers to GIS interoperability. *International Journal of Geographical Information Science*, v. 12, p. 299–314, 1998.

Bleiholder, J.; Naumann, F. Conflict handling strategies in an integrated information system. In: *Proceedings of the International Workshop on Information Integration on the Web (IIWeb)*, 2006.

Bleiholder, J.; Naumann, F. Data fusion. *ACM Computing Survey*, v. 41, n. 1, p. 1–41, 2008.

Boag, S.; Chamberlin, D.; Fernández, M. F.; Florescu, D.; Robie, J.; Siméon, J. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/, 2007.

Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E.; Yergeau, F. *Extensible markup language (XML) 1.0.* World Wide Web Consortium (W3C), http://www.w3.org/TR/REC-xml, 1998.

BUNEMAN, P.; CHAPMAN, A.; CHENEY, J. Provenance management in curated databases. In: *SIGMOD'06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, 2006, p. 539–550.

BUNEMAN, P.; DAVIDSON, S.; FAN, W.; HARA, C.; TAN, W.-C. Keys for XML. In: *WWW'01: Proceedings of the 10th International Conference on World Wide Web*, 2001a, p. 201–210.

BUNEMAN, P.; KHANNA, S.; TAN, W. C. Why and where: A characterization of data provenance. In: *ICDT'01: Proceedings of 8th International Conference on Database Theory*, 2001b, p. 316–330.

CHAN, L. M.; MITCHELL, J. S. *Introduction to the Dewey Decimal Classification*. http://www.oclc.org/dewey/versions/ddc22print/intro.pdf, 2003.

CHAWATHE, S.; GARCIA-MOLINA, H.; HAMMER, J.; IRELAND, K.; PAPAKONSTANTINOU, Y.; ULLMAN, J.; WIDOM, J. The TSIMMIS project: Integration of heterogeneous information sources. In: *In Proceedings of IPSJ Conference*, 1994, p. 7–18.

CLARK, J.; DEROSE, S. XML Path Language (XPath). World Wide Web Consortium (W3C), http://www.w3.org/TR/xpath, 1999.

CNPQ Plataforma Lattes. http://lattes.cnpq.br/conteudo/aplataforma.htm, 2010.

CRUZ, I. F.; XIAO, H. The role of ontologies in data integration. *Journal of Engineering Intelligent Systems*, v. 13, n. 4, p. 245–252, 2005.

DEUTSCH, A.; TANNEN, V. Containment and integrity constraints for XPath. In: *Proceedings of the 8th International Workshop on Knowledge Representation meets Databases*, 2001.

DRAPER, D.; HALEVY, A. Y.; WELD, D. S. The Nimble XML data integration system. In: *Proceedings of the 17th International Conference on Data Engineering*, Washington, DC, USA: IEEE Computer Society, 2001, p. 155–160.

DUSCHKA, O. M.; GENESERETH, M. R. Answering recursive queries using views. In: *PODS'97: Proceedings of the 6th Symposium on Principles of Database Systems*, 1997, p. 109–116.

FALLSIDE, D. C. *XML Schema Part 0: Primer*. World Wide Web Consortium (W3C), http://www.w3.org/TR/xmlschema-0/, 2000.

FRIEDMAN, M.; LEVY, A.; MILLSTEIN, T. Navigational plans for data integration. In: *AAAI'99: Proceedings of the National Conference on Artificial Intelligence*, AAAI Press/The MIT Press, 1999, p. 67–73.

FUXMAN, A.; FAZLI, E.; MILLER, R. J. ConQuer: efficient management of inconsistent databases. In: *SIGMOD'05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, 2005, p. 155–166.

GALHARDAS, H. Data cleaning and transformation using the AJAX framework. In: *Generative and Transformational Techniques in Software Engineering, International Summer School*, 2006, p. 327–343.

GALHARDAS, H.; FLORESCU, D.; SHASHA, D.; SIMON, E. AJAX: an extensible data cleaning tool. *SIGMOD Rec.*, v. 29, n. 2, p. 590, 2000.

GALHARDAS, H.; FLORESCU, D.; SHASHA, D.; SIMON, E.; SAITA, C.-A. Declarative data cleaning: Language, model, and algorithms. In: *Proceedings of 27th International Conference on Very Large Data Bases*, 2001, p. 371–380.

HAAS, L. M.; KOSSMANN, D.; WIMMERS, E. L.; YANG, J. Optimizing queries across diverse data sources. In: *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, p. 276–285.

HAMMER, J.; MCHUGH, J.; GARCIA-MOLINA, H. Semistructured data: The tsimmis experience. In: *Proceedings of the First East-European Symposium on Advances in Databases and Information Systems*, 1997, p. 1–8.

HERNÁNDEZ, M. A.; STOLFO, S. J. The merge/purge problem for large databases. In: *SIGMOD'95: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, 1995, p. 127–138.

HOPCROFT, J. E.; ULLMAN, J. D. *Introduction to automata theory, languages and computation.* Addison-Wesley, 1979.

HYLTON, J. A. *Identifying and merging related bibliographic records.* Relatório Técnico, Massachusetts Institute of Technology, Cambridge, MA, USA, 1996.

JIN, L.; LI, C.; MEHROTRA, S. Efficient record linkage in large data sets. In: *DASFAA'03: Proceedings of the Eighth International Conference on Database Systems for Advanced Applications*, 2003, p. 137.

LEVY, A. Y.; RAJARAMAN, A.; ULLMAN, J. D. Answering queries using limited external query processors (extended abstract). In: *PODS '96: Proceedings of the 15th Symposium on Principles of Database Systems*, New York, NY, USA: ACM, 1996, p. 227–237.

LEY, M. The DBLP computer science bibliography. http://www.informatik.uni-trier.de/~ley/db/, 2010.

MIKLAU, G.; SUCIU, D. Containment and equivalence for a fragment of XPath. *Journal of the ACM*, v. 51, n. 1, p. 2–45, 2004.

MOTRO, A. Multiplex: A formal model for multidatabases and its implementation. In: *4th International Workshop on Next Generation Information Technologies and Systems*, 1999, p. 138–158.

MOTRO, A.; ANOKHIN, P. Utility-based resolution of data inconsistencies. In: *IQIS'04: Proceedings of International Workshop on Information Quality in Information Systems*, 2004, p. 35–43.

MOTRO, A.; ANOKHIN, P. Fusionplex: resolution of data inconsistencies in the integration of heterogeneous information sources. *Information Fusion*, v. 7, n. 2, p. 176–196, 2006.

MOTRO, A.; BERLIN, J.; ANOKHIN, P. Multiplex, Fusionplex, and Autoplex - Three generations of information integration. *SIGMOD Record*, v. 33, n. 4, p. 51–57, 2004.

DO NASCIMENTO, A. M.; HARA, C. S. A model for XML instance level integration. In: *SBBD'08: Proceedings of the 23rd Brazilian Symposium on Databases*, 2008, p. 46–60.

NEVEN, F.; SCHWENTICK, T. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Logical Methods in Computer Science*, v. 2, n. 3, 2006.

PEMBERTON, S.; AUSTIN, D.; AXELSSON, J.; ÇELIK, T.; DOMINIAK, D.; ELENBAAS, H.; EPPERSON, B.; ISHIKAWA, M.; MATSUI, S.; MCCARRON, S.; NAVARRO, A.; PERUVEMBA, S.; RELYEA, R.; SCHNITZENBAUMER, S.; STARK, P. *Document type definitions*. World Wide Web Consortium (W3C), http://www.w3.org/TR/xhtml1/dtds.html, 2002.

POPA, L., ed. *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems*, ACM, 2002.

QUASS, D.; STARKEY, P. Record linkage for genealogical databases. In: *In KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003, p. 40–42.

RAHM, E.; DO, H. H. Data cleaning: Problems and current approaches. *IEEE Bulletin of the Technical Committee on Data Engineering*, v. 23, n. 4, p. 3–13, 2000.

RAMAN, V.; HELLERSTEIN, J. M. Potter's Wheel: An interactive data cleaning system. In: *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, 2001, p. 381–390.

SCHALLEHN, E.; SATTLER, K.-U.; SAAKE, G. Efficient similarity-based operations for data integration. *Data Knowledge Engineering*, v. 48, n. 3, p. 361–387, 2004.

TAN, W. C. Provenance in databases: Past, current, and future. *IEEE Data Engineering Bulletin*, v. 30, n. 4, p. 3–12, 2007.

TATARINOV, I.; VIGLAS, S. D.; BEYER, K.; SHANMUGASUNDARAM, J.; SHEKITA, E.; ZHANG, C. Storing and querying ordered XML using a relational database system. In: *SIGMOD'02: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, 2002, p. 204–215.

WEIS, M.; MANOLESCU, I. Declarative XML data cleaning with XClean. In: *CAiSE'07: Proceedings of the 19th International Conference Advanced Information Systems Engineering*, 2007, p. 96–110.

WINKLER, W. E. *Advanced methods for record linkage.* Relatório Técnico, Statistical Research Division, U.S. Bureau of the Census., 1994.

WOOD, P. T. Containment for XPath fragments under DTD constraints. In: *ICDT*, 2003, p. 297–311.

YAN, T. W.; GARCIA-MOLINA, H. Duplicate removal in information system dissemination. In: *VLDB'95: Proceedings of the 21th International Conference on Very Large Data Bases*, 1995, p. 66–77.